Санкт-Петербургский Государственный Университет

Факультет прикладной математики — процессов управления

Шевелев Александр Александрович

Магистерская диссертация

Применение методов топологического анализа данных при балансировке нагрузки в вычислительных сетях

Направление 010300

«Фундаментальная информатика и информационные технологии»

Магистерская программа «Технологии баз данных»

Руководитель магистерской программы,
кандидат физ.-мат. наук,
доцент Сергеев С. Л.

Научный руководитель,
ст. преподаватель Мозжерина Е. С.

Рецензент,
ведущий разработчик,
ООО «АйТи Либертас» Баранов Д. Е.

Санкт-Петербург

2016

Содержание

Введение

За довольно непродолжительное время количество данных, генерируемых человечеством в единицу времени, невероятно возросло. Буквально каждую минуту генерируется информация сравнимая по объёму с информацией, на создание которой в прошлом требовались десятки лет. Одной из причин такого роста является развитие технологий, в частности, развитие интернета и вычислительных сетей.

Интернет можно рассматривать как совокупностью информационных систем, каждая из которых осуществляет обработку информации. При увеличении объёма данных, которые обрабатывает информационная система, производительности, которую может обеспечить одна вычислительная машина становится недостаточно, что послужило причиной объединения вычислительных машин в вычислительные сети, обеспечивающие работу информационных систем.

Становление вычислительных сетей в качестве основы для работы информационных систем привело к увеличению вычислительных возможностей, доступных к использованию в рамках информационной системы. Однако переход к распределенной архитектуре сопровождается рядом новых проблем и задач, с которыми не приходится сталкиваться в рамках нераспределённой архитектуры. К таким задачам можно отнести: обеспечение согласованности данных в рамках вычислительной сети, обнаружение завершения выполнения задачи, необходимость коммутации пакетов в рамках вычислительной сети и балансировка нагрузки в рамках информационной системы.

Есть основания полагать, что в будущем количество данных, генерируемых человечеством, будет продолжать быстро расти. Этому будут способствовать и многие нововведения из мира информационных технологий, в качестве примера можно указать набирающие всё большую популярность устройства, относящиеся к

классу интернета вещей. Очевидно, что при увеличении количества таких устройств, информационная нагрузка, как на частные локальные сети, так и на вычислительные сети информационных систем, обеспечивающие работу таких устройств, будет возрастать. В условиях увеличивающейся нагрузки работа информационных систем, в том числе и алгоритмы обеспечивающие внутреннюю работу вычислительной сети, должны совершенствоваться и работать более эффективно.

Одним из аспектов требующих наиболее эффективной работы программного комплекса вычислительной сети является пост-обработка и анализ результатов выполнения вычислительной сетью входящих задач. Данная работа нацелена на решение именно этой задачи, а именно на анализ набора данных, описывающих работу вычислительного комплекса, с целью выявления типов выполненных операция для последующей выработки стратегии балансировки нагрузки.

Постановка задачи

Целью данной магистерской диссертации является проверка гипотезы о том, что применение топологических методов анализа данных позволит выбирать более эффективные стратегии балансировки нагрузки в вычислительной сети за счёт выявления дополнительных особенностей, не выявляемых традиционными методами анализа данных.

Для проверки этой гипотезы необходимо осуществить сравнение результатов, полученных с помощью традиционных и топологических методов анализа данных.

Таким образом, необходимо реализовать несколько традиционных методов анализа данных для осуществления автоматического разделения различных типов исполняемых вычислительной сетью задач.

Помимо этого необходимо реализовать несколько топологических методов анализа данных для осуществления разделения типов исполняемых вычислительной сетью задач.

В качестве базового набора данных для анализа будет использован информационный след, записанный вычислительными кластерами компании Google [1]. Данный набор опубликован в неочищенном и частично обфусцированном виде, поэтому перед тем как выполнять его анализ необходимо осуществить предобработку данных.

Глава 1. Теоретические основы. Обзор литературы

Вычислительной сетью называется взаимосвязанная совокупность территориально рассредоточенных систем обработки данных, средств и/или систем связи и передачи данных, обеспечивающая пользователям дистанционный доступ к её ресурсам и коллективное использование этих ресурсов [2]. Для большей конкретики приведённое определение необходимо уточнить. Стоит отметить, что под территориально распределенными системами обработки данных понимаются как физические вычислительные системы, так и виртуальные, представляющие собой части публичных, приватных или гибридных облачных структур типа инфраструктура как сервис. При этом под вычислительной системой в данной работе подразумевается одна или несколько вычислительных машин, физических или виртуальных, гомогенной природы, таким образом, вычислительная система, в приведённом понимании, является наименьшей вычислительной единицей, рассматриваемой в данной работе.

Под балансировкой нагрузки в данной работе понимается распределение требуемых к исполнению вычислительной сетью задач между вычислительными системами так, чтобы задачи были выполнены наиболее оптимальным образом.

Каждая вычислительная сеть, гетерогенной или гомогенной природы, строится с целью выполнения некоторых вычислительных задач, например обеспечение работы пользовательских сервисов или обработка данных. Для наиболее полного и эффективного использования вычислительных ресурсов необходимо использовать некоторую стратегию балансировки поступающих задач.

Предполагается, что стратегия балансировки нагрузки в рамках вычислительной сети строится на основе автоматического анализа собранных данных. В данной работе планируется применить традиционные методы анализа

данных, а затем сравнить их результаты с результатами выполнения анализа данных топологическими методами.

1.1 Балансировка нагрузки

Проблема балансировки нагрузки, в настоящее время, представляет собой достаточно актуальную и нетривиальную задачу в связи с непрекращающимся увеличением вычислительной нагрузки и потоков данных как в рамках глобальной сети интернет, так и в рамках исследовательских сетей. Ярким примером могут послужить колоссальные потоки данных генерируемые большим адронным коллайдером. Вопросы балансировки нагрузки серьёзно обострились и с развитием технологий, в частности с появлением и развитием облачных вычислений. Подтверждение этому можно найти в работах [3], [4] и [5]. Рассмотрим подробно определения, проблемы и классификации приведённые в этих работах.

Балансировка нагрузки — одна из ключевых составляющих эффективного использования распределенных вычислительных сетей [3].

В статье [3] приведена подробная классификация проблем возникающих во время решения задачи балансировки нагрузки в вычислительной сети, помимо этого статья содержит описание и классификацию существующих на текущий момент алгоритмов балансировки нагрузки.

Проблемы, возникающие при балансировке нагрузки в [3] разделены на следующие категории.

1. Физическая распределённость узлов вычислительной сети.
2. Хранение данных/Репликация.
3. Алгоритмическая сложность.
4. Возможное наличие единой точки отказа.

В приведённой статье существующие алгоритмы балансировки нагрузки разделены на два больших класса.

1. Статические алгоритмы балансировки нагрузки, принимающие во внимание только доступность узла вычислительной сети.
2. Динамические алгоритмы балансировки нагрузки учитывающие иные параметры (загруженность узла, пропускную способность сети и т. д.).

В отличие от [3], в [4] приведено подробное определение балансировки нагрузки в вычислительной сети.

Балансировка нагрузки — процесс перераспределения полной нагрузки между узлами вычислительной сети таким образом, чтобы минимизировать время решения входящих задач и максимально использовать ресурсы вычислительной сети [4].

Помимо более подробного определения в [4] приведена классификация существующих подходов к балансировке нагрузки в вычислительной сети, был выделен приведённый ниже список методов.

1. Быстрая, адаптивная балансировка нагрузки.
2. Методы балансировки нагрузки на основе поведения медоносных пчёл.
3. Динамические адаптивные методы балансировки нагрузки для параллельных файловых систем.
4. Балансировка нагрузки в больших многопользовательских виртуальных окружениях.
5. Балансировка нагрузки в динамических структурированных P2P системах.

Кроме классификации в [4] приведено подробное описание параметров, учитываемых каждой из методик и их преимущества и недостатки.

В [5] приведено несколько отличное определение балансировки нагрузки, сделано это с целью определения чуть более узкого класса алгоритмов балансировки нагрузки, а именно балансировки нагрузки в среде облачных вычислений.

Балансировка нагрузки — методика, позволяющая распределить избыточную динамическую нагрузку равномерно по всем узлам входящим в облачную вычислительную сеть. Эта методика используется для достижения наилучшей производительности и наиболее эффективного использования доступных вычислительных ресурсов [5].

В [5] приведены задачи и цели осуществления балансировки нагрузки в облачных окружениях, рассмотрим заявленный список целей осуществления балансировки нагрузки.

1. Повысить доступность сервиса.
2. Повысить удовлетворённость пользователей сервисом.
3. Использовать имеющиеся вычислительные ресурсы наиболее эффективным образом.
4. Минимизировать время выполнения входящих задач и время ожидания результатов их выполнения.
5. Улучшить производительность вычислительной сети.
6. Повысить стабильность информационной системы.
7. Построить вычислительную сеть, устойчивую к возникающим ошибкам.
8. Облегчить поддержку будущих изменений.

Кроме приведённых целей в [5] упоминаются следующие проблемы возникающие при балансировке нагрузки.

1. Расход ресурсов на систему балансировки нагрузки.
2. Пропускная способность.
3. Производительность.
4. Использование ресурсов.
5. Масштабируемость.
6. Время отклика.
7. Устойчивость к ошибкам.

8. Наличие единой точки отказа.

В [5], как и в [3] приводится классификация методов балансировки нагрузки, но по сравнению с [3] она имеет более расширенный характер и иерархическую структуру. Согласно [5] существующие методы балансировки нагрузки в вычислительной сети разделяются на следующие классы.

1. Основанные на текущем состоянии системы.
    1.1. Статические.
    1.2. Динамические.
2. Основанные на начальном состоянии.
    2.1. Инициированные отправителем.
    2.2. Инициированные получателем.
    2.3. Симметричные.

В данной работе в качестве целевого метода балансировки нагрузки предполагается использовать метод балансировки нагрузки основанный на исторической информации, полученной из информационного следа выполнения задач вычислительной сетью. Предполагается, что этот алгоритм будет учитывать как технические параметры (текущая загрузка узлов вычислительной сети, пропускная способность сетевых каналов и т. д.), так и предположения о природе входящей задачи, построенные на основе анализа логов загрузки вычислительной сети.

1.2 Топологические методы анализа данных

Топологические методы анализа данных — достаточно новая ветвь статистического анализа, цель которой состоит в приложении топологии к разработке инструментов изучения независимых от выбранного масштаба, глобальных, нелинейных, геометрических свойств данных. [6]

Комплекс существующих на сегодняшний день способов осуществления топологического анализа данных состоит из растущего множества методов, которые позволяют получить представление о «форме» анализируемых данных. Эти инструменты могут быть наиболее эффективно использованы с целью извлечения глобальных свойств анализируемых данных высокой размерности, которые зачастую недоступны иными способами. [7]

Как справедливо отмечается в [8] методы топологического анализа данных, как и многие классические методы анализа данных решают следующие фундаментальные задачи.

1. Представление многомерного объекта с использованием меньшего количества измерений.
2. Представление глобальной структуры из разрозненного множества точек, которые представляют собой реальные данные.

Одной из наиболее исчерпывающих работ в области топологических методов анализа данных и их применения можно считать [9]. В этой статье приведено подробное описание существующих и применяющихся техник анализа данных с точки зрения их практического применения в задачах машинного обучения. Рассмотрим наиболее интересные методы, которые нашли применение в этой и других работах.

Кластеризация в классическом понимании представляет собой разделение большого набора данных на некоторое количество подмножеств на основе геометрии облака точек, соответствующего изначальному набору данных. Конечная цель кластеризации — разделить облако точек на подмножества таким образом, чтобы каждая из точек была ближе к точкам своего подмножества нежели к точкам какого либо иного подмножества. [9] В рамках статьи [10] Джесси Джонсон описывает принципиально новую методику кластеризации

основанную на понятии «тонкой позиции» для 3-многообразий, которое описано в [11].

Одним из активно развивающихся направлений анализа данных в настоящее время является анализ данных путём восстановления их структуры как комбинации вероятностных распределений. [9] Наиболее используемым методом из этого направления является метод восстановления структуры данных как смеси нормальных распределений или смеси гауссиан. В работе [12] приводится обзор методов реализации и описания смеси гауссиан, а также приводится описание собственной методики восстановления структуры анализируемых данных с помощью смеси нормальных распределений высокой размерности.

Некоторые методы топологического анализа данных направлены на избавление от так называемого и хорошо определённого в [13] «проклятия размерности». Проблема проклятия размерности заключается в том, что при наличии большого количества параметров в анализируемых данных некоторые алгоритмы машинного обучения начинают существенно терять в качестве из-за слишком большого количества размерностей вектора данных [13]. Довольно нетривиальным является и решение данной проблемы, а именно выбор подмножества параметров вектора данных, которые позволят, при условии исключения остальных параметров из вектора данных, не потерять в качестве анализа. Иными словами решение проблемы высокой размерности данных — выделение подмножества наиболее значимых параметров в данных.

Топологические методы анализа данных предлагают новый взгляд на методы уменьшения размерности данных. Так, например, в работе [14] приводится алгоритм отображения шумных данных высокой размерности в пространство меньшей размерности с помощью выпуклого растяжения. Помимо этого приводится доказательство устойчивости работы метода главных компонент в условиях выпуклого растяжения описанного в [14].

Приведённые выше методы топологического анализа данных достаточно близко перекликаются с классическими методами анализа данных, далее в данном обзоре будут рассмотрены методы топологического анализа данных, которые в большей степени соответствуют ранее предложенному определению.

Один из наиболее интересных топологических методов решения задачи уменьшения размерности был предложен в [8] и был назван штрихкодами. В данной работе он используется в совокупности с теорией устойчивых гомологий, описанной в [15]. Более подробное описание применяемого метод приведено в главе 2.

Топологический анализ данных и его методы представляют относительно новое направление как в области математики, так и в области компьютерных наук, однако за тот относительно небольшой период, на протяжении которого они применяются с их помощью были получены довольно значимые научные результаты в самых различных областях.

Существуют опубликованные научные результаты применения топологических методов анализа данных например для построения сенсорных сетей наибольшего покрытия [16] или определения новых видов рака [17]. Существуют и результаты из других областей, включающих в себя фармакологию, финансовые рынки и даже баскетбол.

В данной работе рассмотрены и применены топологические методы анализа данных с целью выявления скрытых особенностей задач, содержащихся в информационном следе вычислительного кластера компании Google.

1.3 Применение методов анализа данных в задачах балансировки нагрузки

Различные методы анализа прочно вошли в сферу обеспечения работы вычислительных сетей. С их помощью разрешаются многие вопросы и проблемы в

широком спектре задач, связанных с работой вычислительных сетей. В данной работе будут рассмотрены результаты применения методов анализа данных для решения задач, связанных с балансировкой нагрузки в вычислительных сетях.

В статье [18] приведено описание программного комплекса осуществляющего балансировку нагрузки в рамках вычислительной сети с применением генетических алгоритмов. Одной из отличительных особенностей данной статьи является то, что описываемый балансировщик нагрузки на основе генетических алгоритмов изначально не имеет представления о структуре вычислительной сети и изучает её в процессе выполнения задач. Минусом такого подхода является провал производительности для нескольких первых задач после изменения структуры вычислительной сети, а плюсом — возможность изменения структуры вычислительной сети без необходимости внесения изменений в балансировщик нагрузки.

В статье [19] рассматривается подход к балансировке нагрузки на основе генетических алгоритмов, использующих в качестве тренировочных данных исторический лог выполнения задач вычислительной сетью. В целом предложенный комплекс балансировщика представляет собой вариацию алгоритма Route [20], подкреплённого генетическими алгоритмами, обученными на историческом логе выполнения задач вычислительной сетью. Результаты экспериментов, приведённых в статье, показали, что подкрепление решения стандартного алгоритма балансировки Route классификатором входящих задач приводят к улучшению результатов балансировки и более эффективному использованию ресурсов вычислительной сети.

Статья [21] представляет несколько иной подход к проблеме балансировки нагрузки. Авторы предлагают генерацию автоматических моделей параллелизации входящих задач на основе методов машинного обучения. При этом сначала обучение, параллелизация и балансировка проходят в рамках одной многоядерной

вычислительной машины, а затем обобщаются для работы в рамках вычислительной сети. Обучаемыми структурами в рамках [21] являются нейронные сети.

Выводы по главе

Разнообразие методов анализа данных, применённых и рассмотренных в рамках задачи эффективной балансировки нагрузки в вычислительной сети, достаточно велико. Наибольший прирост в эффективности и производительности вычислительной сети достигается в случаях, когда в качестве системы балансировки нагрузки используется сочетание из эффективного алгоритма балансировки нагрузки и некоторого алгоритма анализа данных, подкрепляющего решения принимаемые балансировщиком нагрузки через набранное в процессе анализа исторических данных о выполненных вычислительной сетью задачах «понимание» о природе входящих задач.

Топологические методы анализа данных являются перспективным направлением исследований в области извлечения скрытых особенностей и связей из данных. Рассмотренные в этой главе научные результаты показывают состоятельность топологических методов анализа данных в тех областях, в которых их уже успели применить ([22], [23]). Результаты применения этих методов позволяют предположить, что они будут также состоятельны и в рамках иных задач, в частности, задач балансировки нагрузки в вычислительной сети.

Глава 2. Предлагаемые подходы и вычислительные эксперименты

В данной главе приведено описание программного комплекса, в рамках которого проводились вычислительные эксперименты и осуществлялось сравнение классических методов анализа данных с топологическими.

2.1 Подготовка данных

В качестве данных для проведения анализа в данной работе используется информационный след задач, выполненных вычислительной сетью компании Google, описание которого приведено в [1]. Полный набор данных состоит из нескольких частей, приведём их список.

1. Описание вычислительных машин.
    1.1. Машинные события (Machine Events).
    1.2. Характеристики машин (Machine Attributes).
2. Глобальные задачи и вычислительные подзадачи (Jobs and Tasks)
    2.1. Таблица событий глобальных задач (Job events table)
    2.2. Таблица событий вычислительных подзадач (Task events table)
3. Ограничения вычислительных подзадач (Task constraints)
4. Использованные ресурсы (Resource usage)

В рамках данной работы наибольший интерес представляет данные принадлежащие к классу 4 (использованные ресурсы). Стоит отметить, что данные, представленные в этом классе являются агрегированными данными класса 2. Именно данные об использованных ресурсах, согласно описанию приведённому в [1], содержат наиболее полную и неизбыточную информацию о природе исполняемых вычислительным кластером задач, для осуществления анализа данных в рамках данной работы используются именно они. Помимо этого в

процессе анализа данных не будет учитываться информация о глобальных задачах, так как они представляют собой набор вычислительных подзадач, которые и являются исполняемыми единицами. При этом информация о глобальных задачах обязательна к использованию в рамках алгоритма балансировки нагрузки, так как некоторые глобальные задачи могут иметь приоритет. В виду того, что анализ данных будет проводится с использованием данных о вычислительных подзадачах, далее в работе именно они будут именоваться задачами.

Рассмотрим доступные в информационном следе данные о ресурсах, использованных задачами. Все доступные агрегированные данные о ресурсах (принадлежащие к классу 4) разделены на 500 примерно одинаковых частей, представляющих собой наборы из примерно 2,5 миллионов 20-мерных векторов, каждый из которых представляет собой одну выполненную кластером Google задачу.

Для каждой из задач, согласно [1], заданы следующие параметры:

1. Время начала периода замеров.
2. Время окончания периода замеров.
3. ID глобальной вычислительной задачи.
4. Глобальный индекс задачи.
5. ID вычислительной машины.
6. Среднее использование CPU.
7. Обычное использование памяти (ОЗУ).
8. Выделенная для задачи память (ОЗУ).
9. Использованные страниц неразмеченной кеш-памяти.
10. Общее использование кеш-памяти.
11. Максимальное использование памяти (ОЗУ).
12. Среднее время использования дисковой памяти.
13. Средний объем использованной дисковой памяти.

14. Максимальное использование CPU.
15. Максимальное время использования дисковой памяти.
16. Циклы на инструкцию (CPI).
17. Доступов к памяти на инструкцию (MAI).
18. Отношение собранных по задаче измерений к ожидаемому количеству измерений по задаче.
19. Тип применённой агрегации.
20. Глобальное среднее использование CPU (среднее использование CPU во время случайного отрезка времени длиной 1 секунда, выбранного из периода проведения замеров)

В Таблице 1 приведён пример данных, анализируемых в рамках данной работы.

Очевидно, что в рамках анализа данных об использованных ресурсах, некоторые из этих данных не будут рассматриваться. В частности перед осуществлением анализа данных будут удалены следующие параметры, не несущие смысловой нагрузки с точки зрения анализа входящих задач: время начала и окончания периода замеров, ID глобальной задачи, индекс задачи, ID машины, отношение собранных по задаче измерений к ожидаемому количеству измерений по задаче, тип агрегации и глобальное среднее использование CPU. Таким образом при анализе каждая из задач будет представлена 12 мерным вектором.

В случае, если данные для некоторых задач окажутся неполными — отсутствующие значения будут заменены нулями.

Помимо этого, несложно отметить, что многие доступные данные в своих значениях отличаются на порядки, поэтому проведём их нормализацию. Нормализация проводится с помощью класса *MinMaxScaler* пакета *scikit.preprocessing*, который приводит каждый столбец данных к виду более или

менее похожему на стандартное нормальное распределение с математическим ожиданием равным нулю и дисперсией равной единице.

Полный набор данных для анализа достаточно велик и занимает около 40 гигабайтов в сжатом виде. Представление о структуре анализируемых данных можно получить не выполняя полный анализ всего объёма представленных данных, а выполнив анализ некоторой выборки из всего объёма доступных данных.

В целях осуществления анализа в рамках данной работы из общего объёма данных были составлены следующие выборки: 10 выборок объёма 0,1% от всего набора данных (примерно 1250000 задач каждая), 10 выборок объёма 1% от всего набора данных (примерно 12500000 задач каждая).

Выборки были составлены следующим образом: из каждой из 500 различных частей составляющих полный набор данных для каждой из упомянутых 20 выборок были выбраны 0,1% и 1% данных. Попадание каждой задачи из каждой 1/500 части полного объёма данных в одну из финальных выборок совпадает с вероятностью попадания в одну из финальных выборок всех остальных задач из соответствующей 1/500.

Полученные таким образом выборки с большой вероятностью репрезентативны. Над каждой из этих выборок будут проведены операции анализа данных, а полученные результаты будут усреднены для выборок составляющих 0.1% и 1%, а затем будет проведено их сравнение между собой.

2.2 Кластеризация методом k-средних

В качестве методологии анализа данных традиционными способами воспользуемся методологией описанной в статье [24]. В ней анализируется старая, менее объёмная версия лога выполненных задач кластера Google, анализ, в отличие от данной работы проводится для глобальных задач.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	600000000	900000000	3418309	0	4155527081	0.001562	0.06787	0.07568	0.001156	0.001503	0.06787	2.861e-06	0.0001869	0.03967	0.0003567	2.445	0.007243	0	1	0.0
1	600000000	900000000	3418309	1	329150663	0.001568	0.06787	0.07556	0.0003195	0.0007	0.06787	5.722e-06	0.0001879	0.03302	0.0009289	2.1	0.005791	0	1	0.0
2	600000000	900000000	3418314	0	3938719206	0.0003071	0.08044	0.09521	0.0002823	0.0006704	0.08044	4.768e-06	0.0001841	0.02377	0.0007858	5.588	0.0208	0	1	0.0
3	600000000	900000000	3418314	1	351618647	0.0003004	0.08044	0.09521	0.0005369	0.0008698	0.08044	9.537e-06	0.0001831	0.007919	0.002285	5.198	0.02038	0	1	0.0
4	600000000	900000000	3418319	0	431052910	0.0004606	0.07715	0.0874	0.0006247	0.0008516	0.07715	1.907e-06	0.0002041	0.005112	0.0002146	2.937	0.009449	0	1	0.0
5	600000000	900000000	3418319	1	257348783	0.0005188	0.07678	0.0874	0.0003376	0.000721	0.07678	6.676e-06	0.0002041	0.01508	0.0009289	3.055	0.01007	0	1	0.0
6	600000000	900000000	3418324	0	5655258253	0.0003805	0.08118	0.08936	0.0004072	0.0007315	0.08118	5.722e-06	0.0001974	0.01624	0.0006428	3.88	0.01376	0	1	0.0
7	600000000	900000000	3418324	1	3550322224	0.0003271	0.08057	0.08936	0.0002928	0.0005913	0.08069	1.907e-06	0.0001974	0.01599	7.153e-05	4.088	0.01569	0	1	0.0
8	600000000	900000000	3418329	0	1303745	0.002518	0.09106	0.1011	0.0004482	0.0007792	0.09106	2.861e-06	0.0003576	0.02121	0.0002146	2.074	0.005869	0	1	0.0
9	600000000	900000000	3418329	1	3894543095	0.002705	0.09082	0.1001	0.0004292	0.0007105	0.09094	6.676e-06	0.0003576	0.01489	0.0009995	2.215	0.006107	0	1	0.0
10	600000000	900000000	3418329	2	336025676	0.002533	0.09094	0.1008	0.0003891	0.0008554	0.09106	2.861e-06	0.0003576	0.02972	0.0003567	2.077	0.00604	0	1	0.0
11	600000000	900000000	3418334	0	3405236527	0.001171	0.08496	0.09521	0.0003309	0.0006666	0.08496	4.768e-06	0.0002155	0.02145	0.0004997	2.489	0.007298	0	1	0.0
12	600000000	900000000	3418334	1	431081448	0.00119	0.08472	0.09521	0.0002127	0.0004034	0.08484	2.861e-06	0.0002165	0.02258	0.0004282	2.257	0.006691	0	1	0.0
13	600000000	900000000	3418339	0	84899647	0.03418	0.08752	0.09839	0.001888	0.002773	0.08887	0.0002737	0.0002117	0.103	0.05786	1.859	0.003949	0	1	0.0
14	600000000	900000000	3418339	1	1268205	0.03528	0.0874	0.099	0.0006256	0.001501	0.08862	0.0002241	0.0002146	0.1229	0.02271	1.976	0.004837	0	1	0.0
15	600000000	900000000	3418339	2	7753127	0.03644	0.08704	0.09912	0.0007992	0.001247	0.08862	0.0002422	0.0003519	0.1141	0.03036	1.947	0.004163	0	1	0.0
16	600000000	900000000	3418339	3	351621284	0.03845	0.08728	0.09912	0.0007133	0.001493	0.0885	0.0001917	0.0002146	0.1082	0.02972	1.913	0.004048	0	1	0.0
17	600000000	900000000	3418339	4	317488701	0.0379	0.08716	0.09912	0.0008802	0.00149	0.08838	7.725e-05	0.0003586	0.1155	0.004997	2.047	0.004314	0	1	0.0
18	600000000	900000000	3418339	5	2595183881	0.03406	0.08704	0.09912	0.001905	0.002449	0.08838	7.629e-05	0.000349	0.1353	0.01401	1.82	0.003891	0	1	0.0
19	600000000	900000000	3418339	6	621588868	0.03699	0.08704	0.09912	0.0008173	0.001331	0.0885	0.0001936	0.0003576	0.1348	0.0354	1.894	0.003909	0	1	0.0

Таблица 1: Пример анализируемых данных.

Так же как и в [24], в данной работе для кластеризации задач будет использован известный алгоритм k-средних, а именно его реализация из пакета scikit-learn. Эта реализация использует классический алгоритм k-means описанный Ллойдом в 1957 году, а затем опубликованный в [25].

Данный алгоритм будет запускаться для возрастающих от 1 значений k_i (количество кластеров). Продолжать увеличение количества кластеров будем до тех пор, пока при переходе от некоторого значения k_n к следующему значению k_{n+1} не начнёт проявляться значительное смешение элементов непересекающихся до этого кластеров. Таким образом, получим иерархию кластеров. Примерно такая же методика была применена и в [24]. Найденное таким образом число k_n будет представлять собой количество хорошо разделимых типов задач выполняемых кластером Google. После выполнения этой операции можно перейти к рассмотрению свойств задач, принадлежащих к каждому из кластеров и определению характеристик каждого из выделенных типов задач.

2.3 Кластеризация с помощью смеси гауссиан

После завершения работы с k-средних попробуем выполнить кластеризацию иным методом, будем использовать методом смеси гауссиан. В отличие от k-средних метод кластеризации с помощью смеси гауссиан не требует задания количества кластеров.

Для реализации данного метода воспользуемся классификатором *DPGMM* из пакета *sklearn.mixture*. Данный классификатор реализует модель смеси гауссиан с использованием иерархического процесса Дирихле. [25]

Согласно документации *sklearn* о классификаторах реализующих модели смеси гауссиан [26], классификатор DPGMM не требует от пользователя входных параметров кроме примерной верхней границы количества компонентов смеси

гауссиан (n) и параметра α , представляющий параметр концетрации процесса Дирихле. При этом, в отличие от классической реализации смеси гауссиан на основе EM-алгоритма [27], модель смеси гауссиан с использованием процесса Дирихле способна автоматически ограничивать количество гауссиан используемых в смеси и, за счёт этого, более точно определять кластеры.

Постепенно увеличивая параметры принимаемые на вход классификатором *DPGMM* добьёмся такого сочетания параметров n и α , при котором верно, что если увеличивать любой из этих параметров — количество найденных классификатором кластеров не изменится.

После завершения подбора параметров рассмотрим получившиеся кластеры и определим свойства задач, принадлежащих к каждому из выделенных кластеров.

2.4 Выявление различных типов задач с помощью устойчивых гомологий

Не так давно появившаяся теория устойчивых гомологий строится на представлении облака точек данных как некоторого набора симплициальных комплексов. Определение устойчивой гомологии дано в статье [28] и звучит следующим образом. Устойчивая гомология — алгебраический инструмент для измерения топологических свойств форм и функций [28].

В статье [29] прекрасно описаны как основные идеи, на которых строится алгоритмы кластеризации с использованием устойчивых гомологий, так и пример такого алгоритма. Рассмотрим основные идеи, на которых строится алгоритм кластеризации с использованием устойчивых гомологий.

1. Должно существовать разбиение облака точек P на кластеры, $\rho=\{\rho_{1,}\rho_{2,..}\rho_{m}\}$ такое, что точки, принадлежащие ρ_i примерно соответствуют выпуклому d - мерному политопу.

2. Пересечение одного из этих политопов с многообразием M , из которого было выбрано облако точек данных P , либо будет представлять собой простую гомотопию, либо быть пустым. Если такое множество оказывается пустым, то это может означать, что точки данных в кластере, соответствующем рассматриваемому политопу, являются шумом, и соответствующее ρ_i может быть удалено из разбиения ρ без потерь.
3. Остающиеся политопы находятся очень близко друг к другу и соответствуют сильно связанным частям рассматриваемого многообразия.
4. Немного раздувая политопы и рассматривая их объединения мы получим некоторое многообразие M' которое гомотопически эквивалентно рассматриваемому ранее многообразию M .

Основываясь на этих принципах возможно выделить сильно связанные компоненты облака точек изначальных данных, применив методику построения штрихкодов, описанную в [8]. Принцип построения штрихкодов заключается в визуализации процесса раздувания политопов и чисел Бетти, соответствующих рассматриваемому многообразию, таким образом, чтобы наиболее длинный непустой интервал соответствовал некоторому k -тому числу Бетти, при этом k представит собой количество связных компонент в рассматриваемом многообразии. Каждая из связных компонент в таком случае будет представлять отдельный тип задач из рассматриваемого набора данных.

В качестве практического инструмента будем использовать пакет TDA, входящий в публичный репозиторий пакетов языка R и позволяющий выполнить описанные операции.

После нахождения параметра k рассмотрим получившиеся типы задач и соответствующие им политопы, определим свойства задач, принадлежащих к каждой из связных компонент многообразия.

2.5 Алгоритм Mapper

Второй из применяемых в данной работе топологических методов работы с данными — алгоритм Mapper, описанный в статье [29]. В данной работе применяется его статистическая реализация, которая основана на идее о том, что кластеризация может быть быть представлена как статистическая версия геометрического представления о разделении пространства на связные компоненты. [29]

По сути, алгоритм Mapper предоставляет инструментарий для визуализации и проекции данных высокой размерности в двух- и трёхмерном пространстве, в виде связных компонент, что позволяет затем подобрать комбинации параметров необходимые для наиболее точного представления компонент сильной связности в облаке рассматриваемых данных.

В качестве практического инструмента рассмотрим библиотеку *KeplerMapper*, строящую проекции с помощью применения алгоритма mapper и выявляющую компоненты сильной связности в облаке данных с помощью алгоритма *DBSCAN* из пакета *sklearn.clustering*.

Выводы по главе

После получения результатов работы алгоритмов, описанных в этой главе проводится сравнение результатов их работы. Целью данной работы является определение применимости топологических методов анализа данных в области балансировки нагрузки в вычислительных сетях, соответственно при сравнении результатов работы описанных алгоритмов будут разрешены следующие аспекты полученных результатов.

1. Качество выделения типов задач, сколько типов задач было выделено каждым из алгоритмов, насколько эти типы действительно разделимы и какими свойствами они обладают.

2. Эффективность работы каждого из алгоритмов, затраченное на обработку данных время и ресурсы.
3. Удобство использования результатов работы описанных алгоритмов. Предполагается, что на основе данных полученных с помощью описанных алгоритмов будет работать механизм подкрепления решений принимаемых алгоритмом балансировки нагрузки.
4. Адаптивность, насколько сложно будет адаптировать описанные алгоритмы в условиях динамически изменяющегося набора данных.

После рассмотрения полученных результатов согласно этим критериям определим применимость топологических методов анализа данных в области балансировки нагрузки в вычислительных сетях и построим рекомендации по применению рассмотренных методов.

Глава 3. Реализация и результаты

В данной главе рассматривается процесс реализации алгоритмов, описанных в главе 2 и полученные результаты их выполнения. Некоторые из предложенных алгоритмов нуждаются в тщательном выборе входных параметров, в данной главе приведено описание процесса и результатов подбора входных параметров для требующих этого алгоритмов.

3.1 Подготовка данных

Для получения наборов данных, анализ которых будет осуществляться, используется пакет *pandas*. Выдержки из реализации на языке *python* приведены в приложении 3.

После создания выборок и первичной фильтрации данных, при которой были отброшены некоторые данные не представляющие интереса в процессе анализа, процесс подготовки данных переходит на следующий этап, а именно, масштабирование данных и дополнительная фильтрация.

В качестве алгоритма масштабирования данных был выбран *MinMaxScaler* из пакета *sklearn.preprocessing*. Идея этого алгоритма состоит в том, что каждый из признаков данных масштабируется таким образом, чтобы все присутствующие в изначальных данных значения этого признака попадали в отрезок [0, 1].

После того, как было выполнено масштабирование, данные были визуализированы на попарных графиках с целью построения предположений о наличии некоторых зависимостей между парами признаков данных и дальнейшей фильтрации. Полная визуализация не будет приведена в силу её огромных размеров, приведём лишь некоторые графики представляющие наибольший интерес.

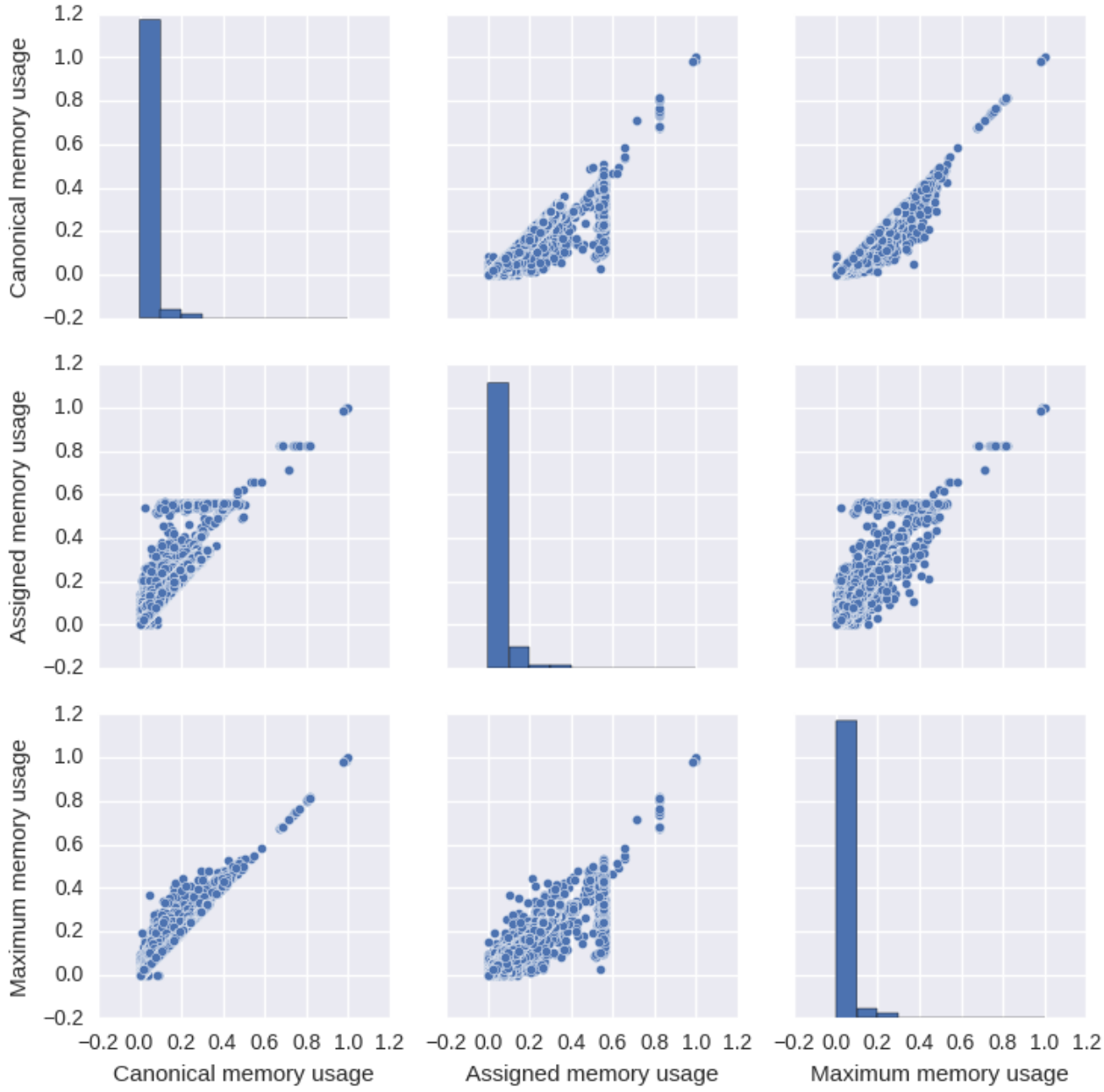

Рисунок 1: Попарные графики признаков, отвечающих за использование ОЗУ.

На рисунке 1 явно прослеживается попарная линейная зависимость между признаками, отвечающими за использование ОЗУ. Некоторое облако вертикальных точек в графиках второго столбца, как оказалось, является не более чем отображением данных, которые не присутствовали в начальной выборке и были заменены нулями.

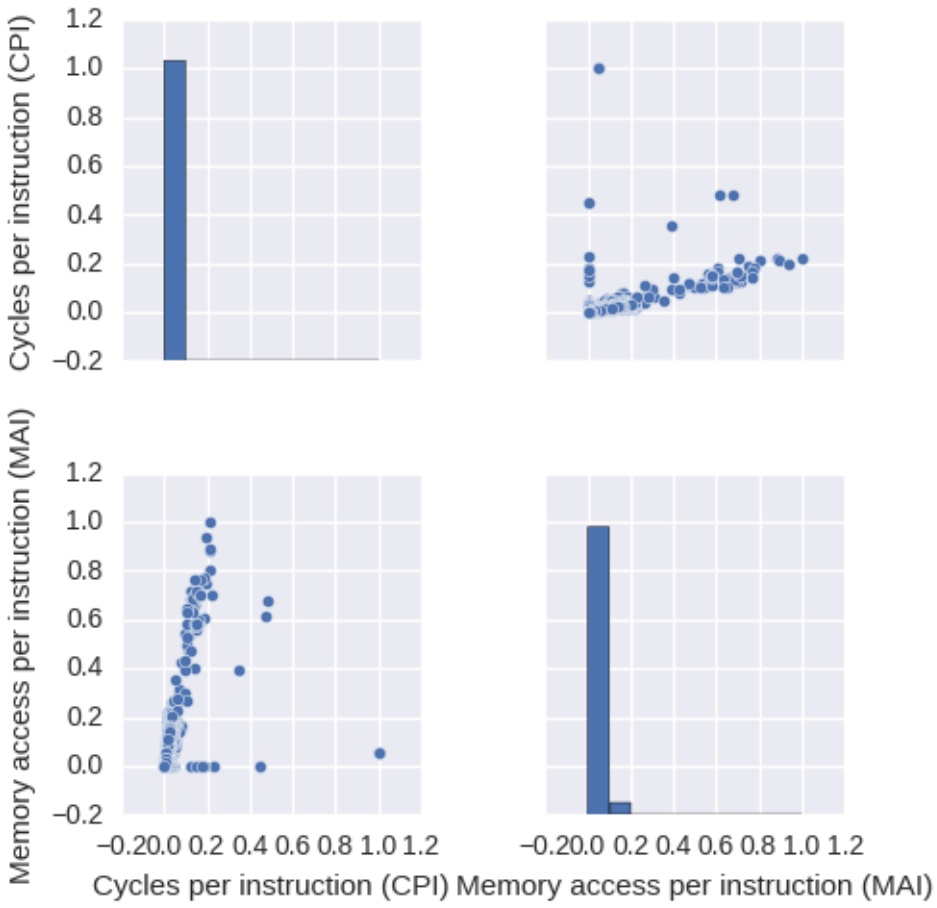

Рисунок 2: Попарный график CPI и MPI.

На рисунке 2 явно видна линейная зависимость между параметрами CPI и MPI, исключением из которой служат лишь некоторые выбросы.

Как видно из рисунка 3, признаки отвечающие за среднее и максимальное использование CPU — не являются линейно зависимыми, это же относится к среднему времени использования дисковой памяти (рисунок 4). В случае же со средним и максимальным временем использования дисковой памяти (рисунок 5), на основе попарной визуализации можно предположить, что между ними может существовать некоторая зависимость, однако такая зависимость явно не является линейной.

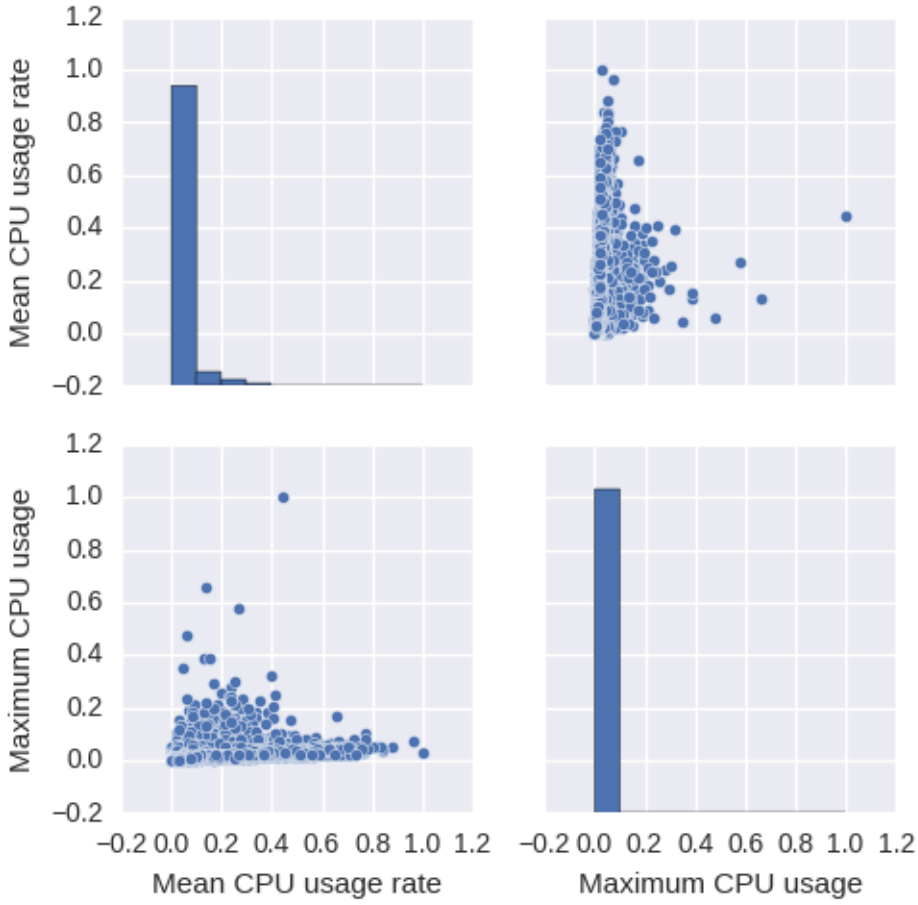

Рисунок 3: Попарные графики признаков, отвечающих за среднее использование CPU и максимальное использование CPU.

Как видно из приведённых графиков, между некоторыми признаками рассматриваемого набора данных прослеживаются явные линейные зависимости. Такие признаки при дальнейшем анализе не представляют интереса в полном объёме, так как являются тривиально выразимыми через меньшее количество признаков, таким образом с целью экономии памяти и времени, оставим в наборе данных девять линейно независимых признаков.

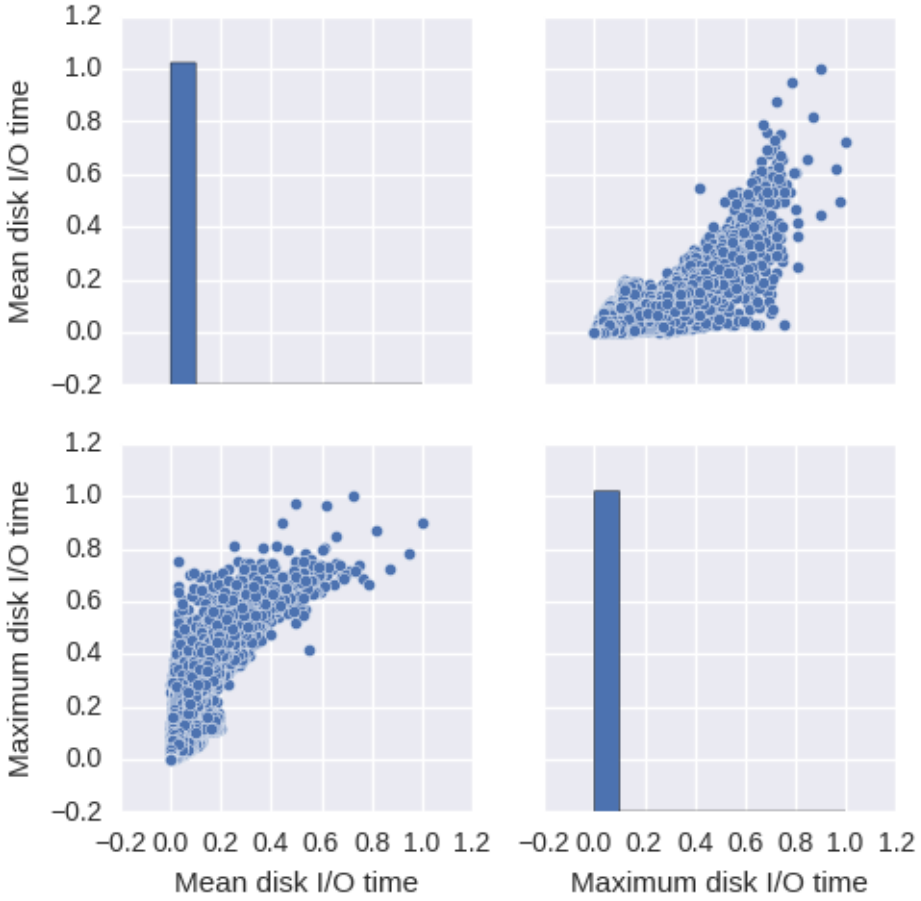

Рисунок 4: Попарные графики среднего и максимального использования дисковой памяти.

1. Среднее использование CPU.
2. Выделенная для задачи память (ОЗУ).
3. Использованные страницы неразмеченной кеш-памяти.
4. Общее использование кеш-памяти.
5. Среднее время использования дисковой памяти.
6. Средний объем использованной дисковой памяти.
7. Максимальное использование CPU.
8. Максимальное время использования дисковой памяти.
9. CPI.

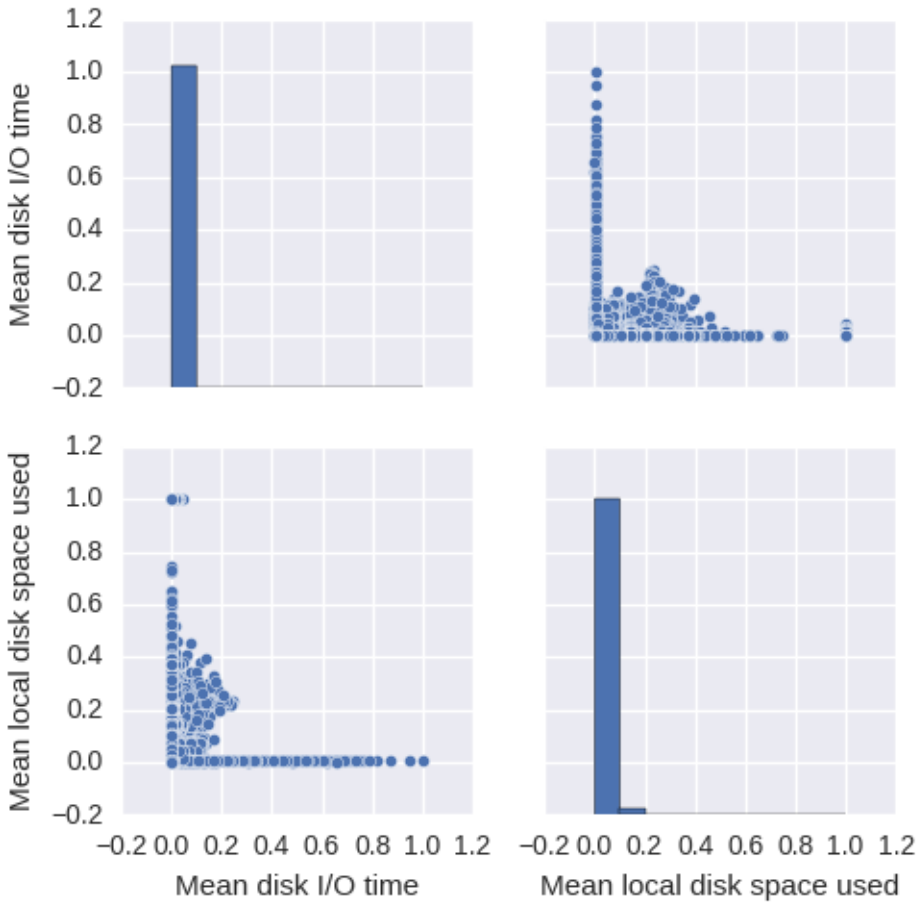

Рисунок 5: Попарные графики среднего времени и среднего объёма использованной дисковой памяти.

3.2 Реализация алгоритма кластеризации k-средних, подбор параметров и результаты

В качестве базового алгоритма кластеризации был выбран алгоритм k-средних, а именно его реализация в пакете *sklearn.clustering*. В качестве входного параметра этот алгоритм принимает количество кластеров, на которые необходимо разделить входные данные.

Подбор необходимого количества кластеров — достаточно нетривиальная задача, в рамках данной работы она решается с использованием метода силуэтов, описанного в работе [30]. Суть этого метода заключается в количественной оценке

качества получившихся кластеров на основе схожести содержащихся в них элементов.

Кластеризация методом k-средних проводилась для разного количества кластеров (от 2 до 8), затем строились силуэты этих кластеров и проводился анализ их содержимого. Результаты этого анализа можно наблюдать на рисунках ниже.

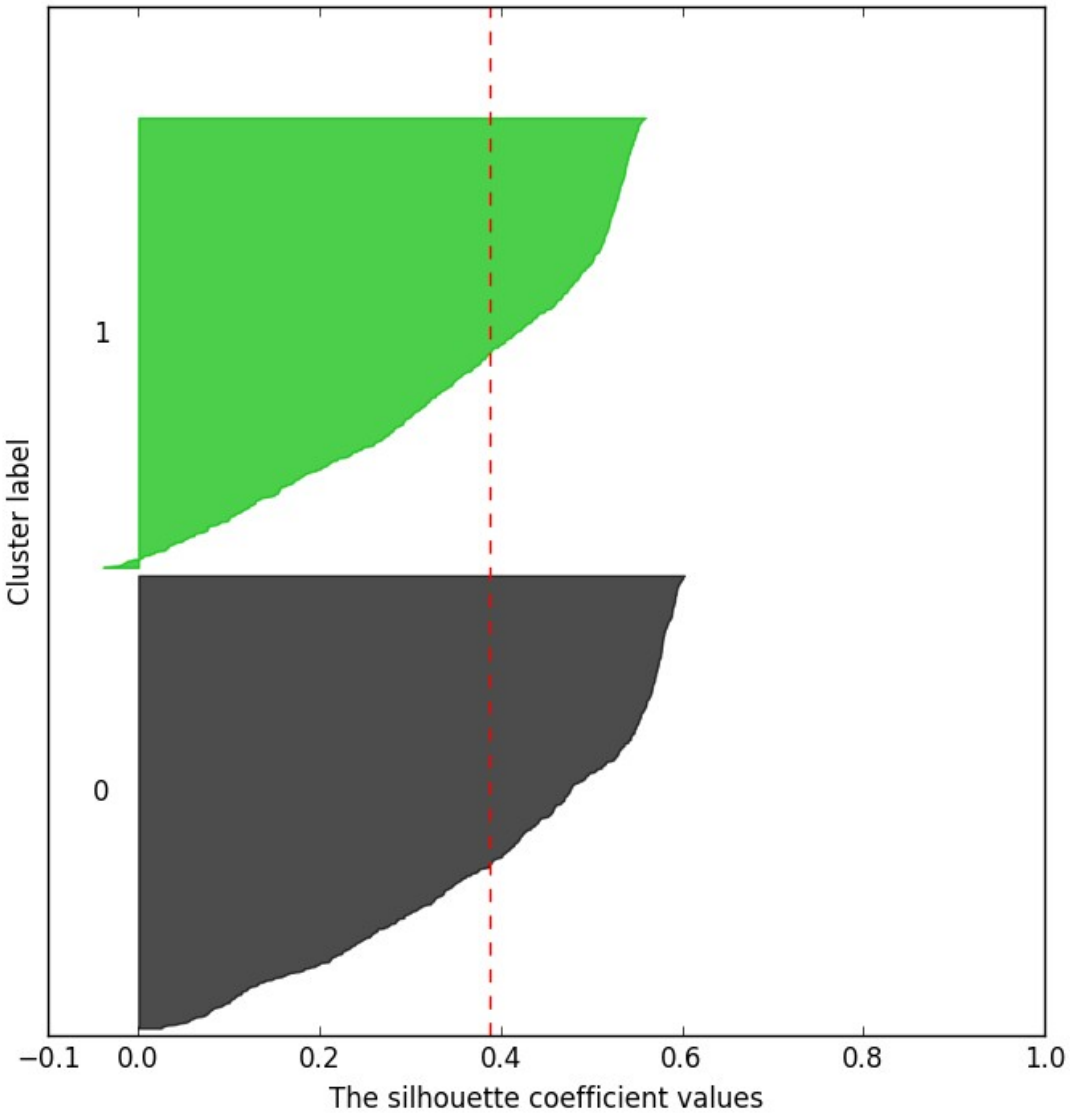

Рисунок 6: Силуэты кластеров при количестве кластеров 2.

Наиболее хорошие результаты кластеризации достигаются при двух значениях количества кластеров: 2 и 4, ниже приведены графики силуэтов кластеров при этих значениях.

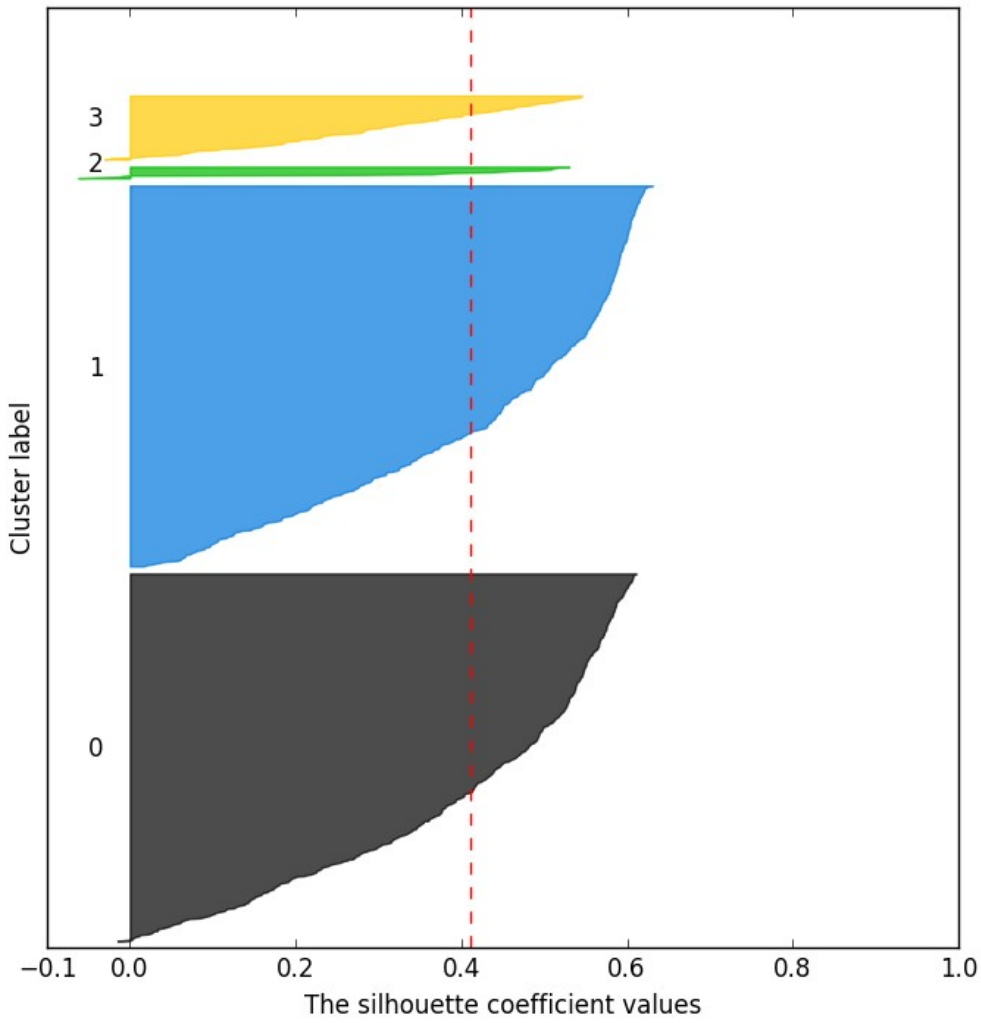

Рисунок 7: Силуэты кластеров при количестве кластеров 4.

При таких значениях входного параметра каждый из кластеров получает коэффициент силуэта превосходящий среднее значение коэффициента силуэта по выборке, что позволяет сделать вывод о том, что оптимальное количество кластеров для рассматриваемой выборки — два или четыре. Однако при выборе

количества кластеров, равного четырём, кластер, представленный на графике цифрой 2 отделяется неявно и при нескольких повторениях процедуры кластеризации элементы из него перемешиваются с элементами из кластеров 3 и 1, что позволяет сделать вывод о недостаточно хорошей разделимости данных по четырём кластерам.

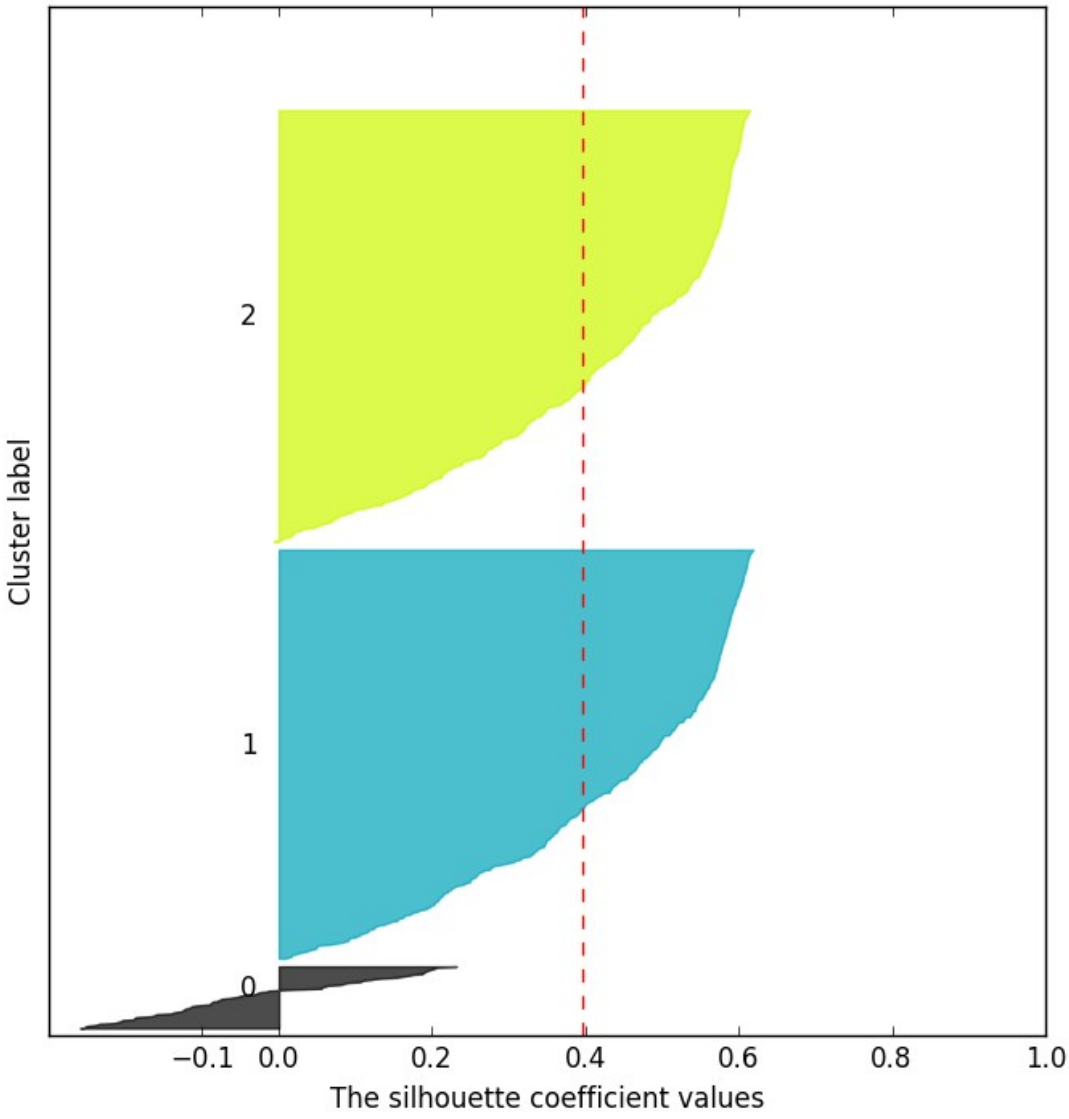

Рисунок 8: Силуэты кластеров при количестве кластеров 3.

Если рассматривать случай разделения набора данных на три кластера, то видно, что при отделении трёх кластеров и последующем анализе получившихся кластеров методом силуэтов, явно разделёнными всегда оказывались два кластера, а третий просто оттягивал на себя некоторые элементы хорошо отделённых кластеров и получал коэффициент силуэта ниже среднего.

Таким образом оптимальное количество кластеров, на которые можно разделить рассматриваемую выборку данных — два. К каждому из этих кластеров относятся задачи обладающие определёнными свойствами, которые рассмотрены в выводах к текущей главе.

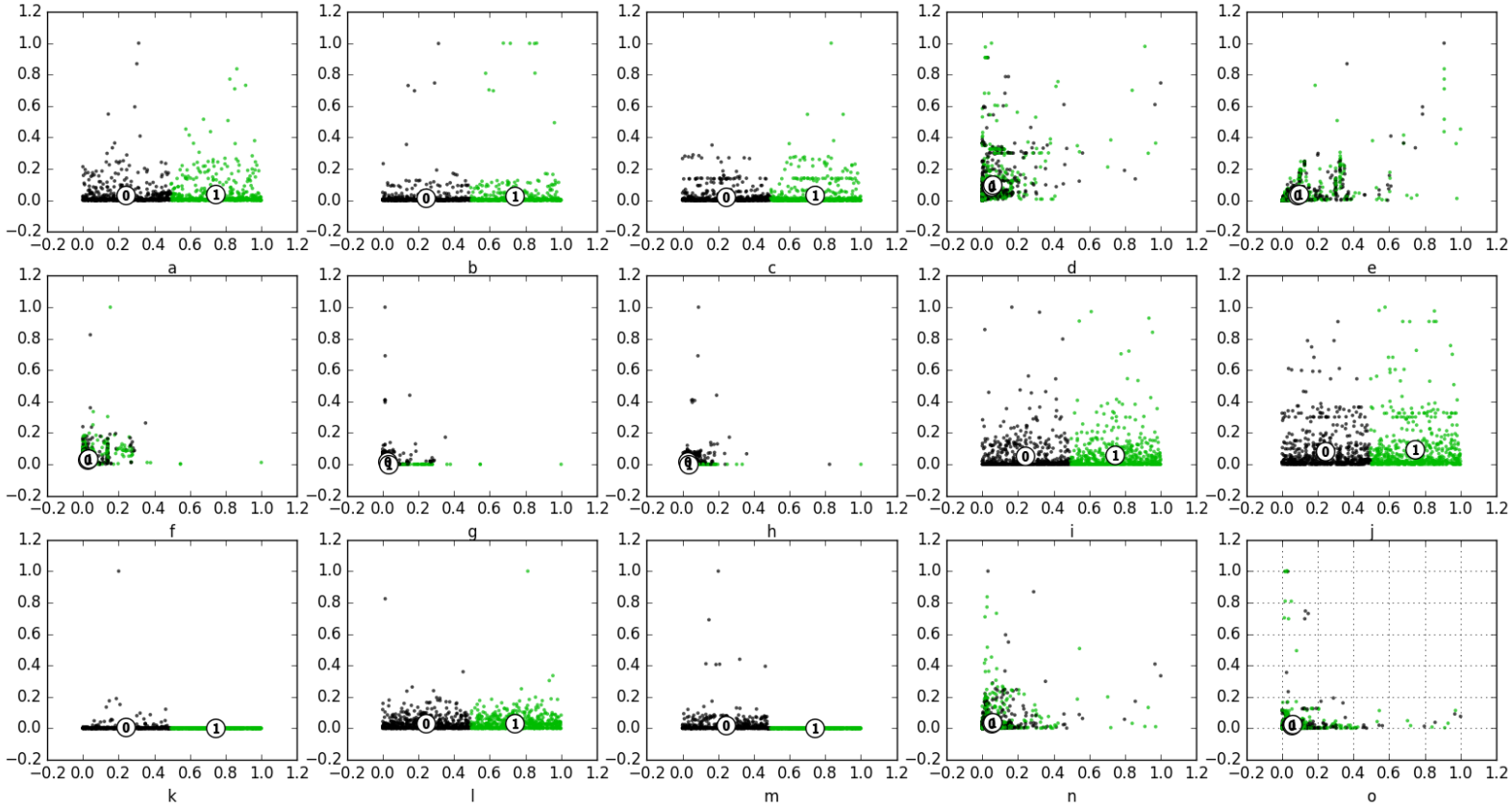

Рисунок 9: Разделение выборки данных на два кластера в проекциях относительно различных характеристик.

3.3 Анализ данных с использованием смеси гауссиан с процессом Дирихле

В качестве реализации модели смеси гауссиан с процессом Дирихле в данной работе используется модуль *DPGMM* из пакета *sklearn.mixture*. Данный модуль обладает существенным недостатком, из-за ошибки в исходном коде модель смеси гауссиан с использованием процесса Дирихле не может быть использована для поиска кластеров в хорошо масштабированных данных. Результатом работы модели в таком случае всегда будет один кластер. Поэтому для работы с этой моделью используем немасштабированный набор данных.

Входным параметром влияющим на количество найденных моделью кластеров является параметр α, представляющий собой величину, характеризующую скорость концентрации процесса Дирихле. Параметр α показывает пороговое количество элементов, начиная с которого процесс Дирихле вероятнее всего отнесёт рассматриваемый элемент к текущему кластеру, нежели начнёт создавать новый кластер. [31]

В процессе поиска оптимального значения параметра α будем присваивать ему значения из набора [0.01, 0.1, 1, 10, 100] и искать кластеры в рассматриваемом наборе данных. Качество кластеризации будем оценивать, как и в случае с k-средних с помощью метода силуэтов. Значение параметра α, при котором найденные кластеры будут наилучшими, будем считать оптимальным.

В процессе изменения подбора параметра α вместе с увеличением параметра увеличивалось и количество кластеров, найденных моделью. При этом количество кластеров увеличивалось от 3 до 8.

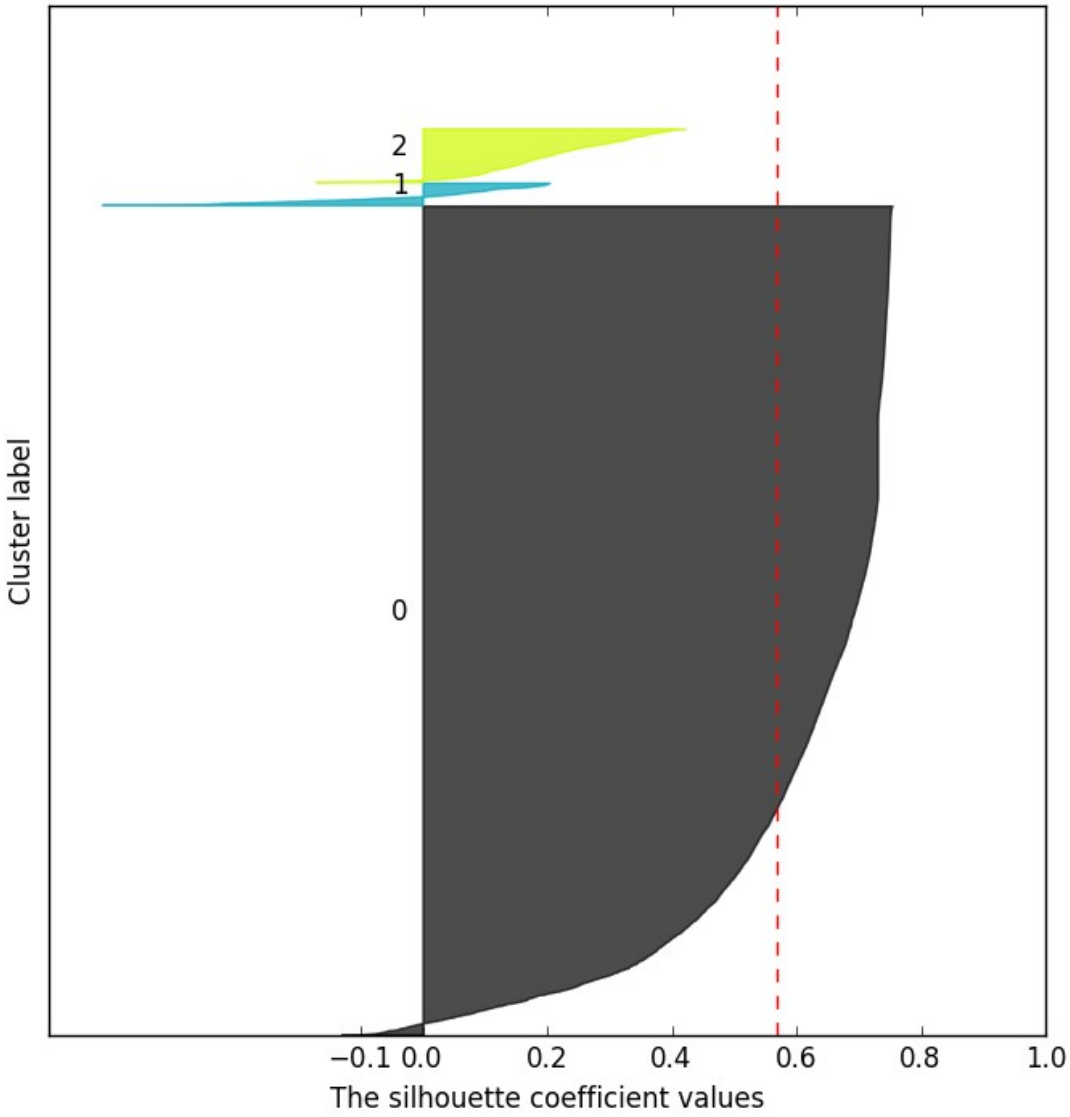

Рисунок 10: Силуэты для малого значения параметра альфа, найдено 3 кластера, однако почти все данные находятся в одном из них.

Результаты поиска силуэтов для различного количества кластеров приведены на рисунках. Не сложно заметить важную особенность: модель смеси гауссиан с процессом Дирихле хоть и выделяет в рассматриваемых данных несколько кластеров, однако большую часть данных всё равно пытается отнести к одному из них.

На основе полученных результатов можно сделать вывод о том, что модель смеси гауссиан с процессом Дирихле не подходит для анализа подобных наборов данных.

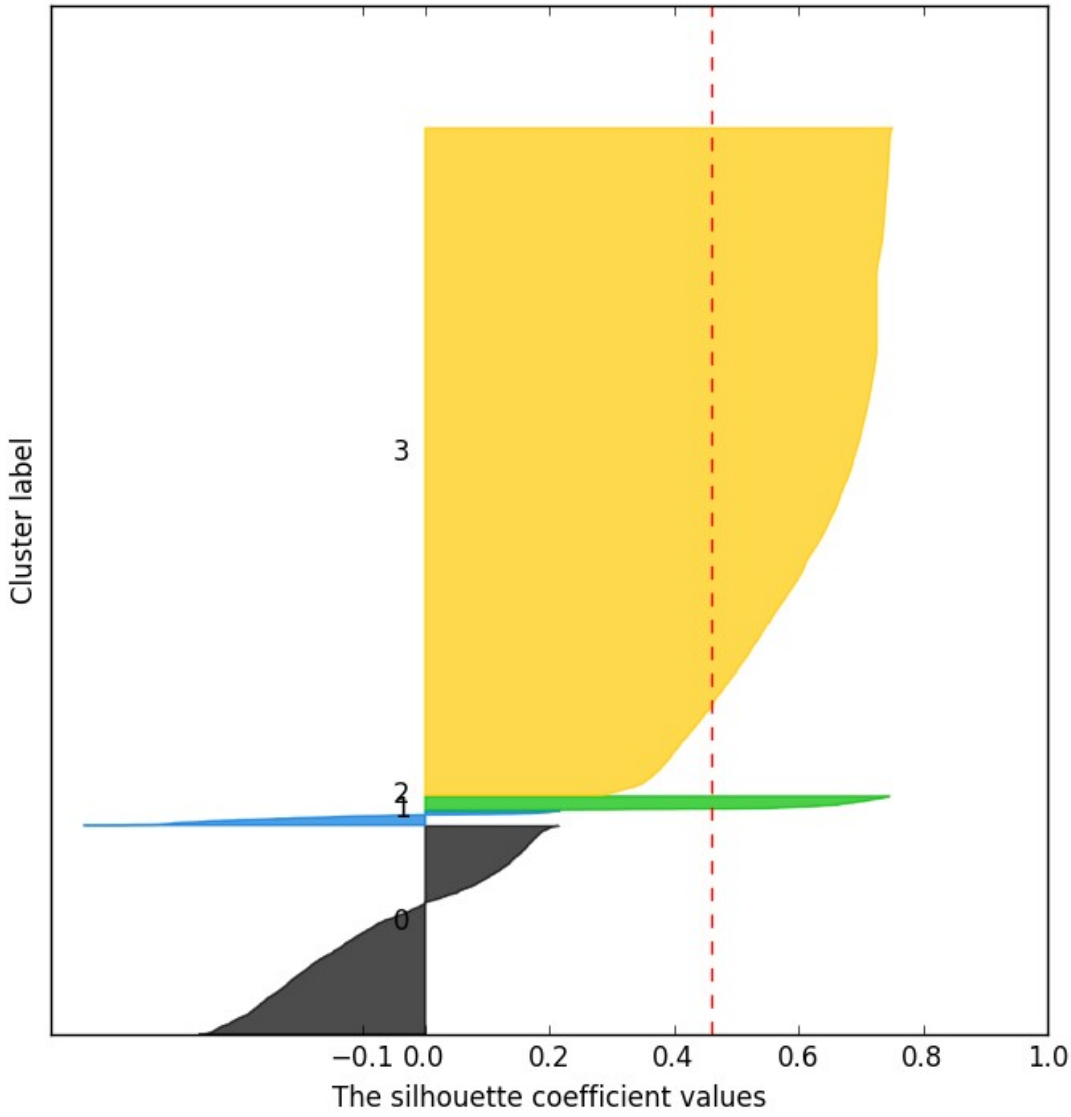

Рисунок 11: Для $\alpha=1$ *найдено 4 кластера, два из них имеют коэффициенты силуэтов выше среднего, однако большая часть данных всё ещё принадлежит одному кластеру.*

3.4 Реализация выявления типов задач с помощью устойчивых гомологий

В качестве библиотеки для поиска устойчивых гомологий в данной работе был использован пакет *TDA*, входящий в публичный репозиторий пакетов языка R. В этот пакет входит широкий инструментарий для анализа данных топологическими методами, полное описаний всех доступных пакетов этого репозитория можно найти в статье [32].

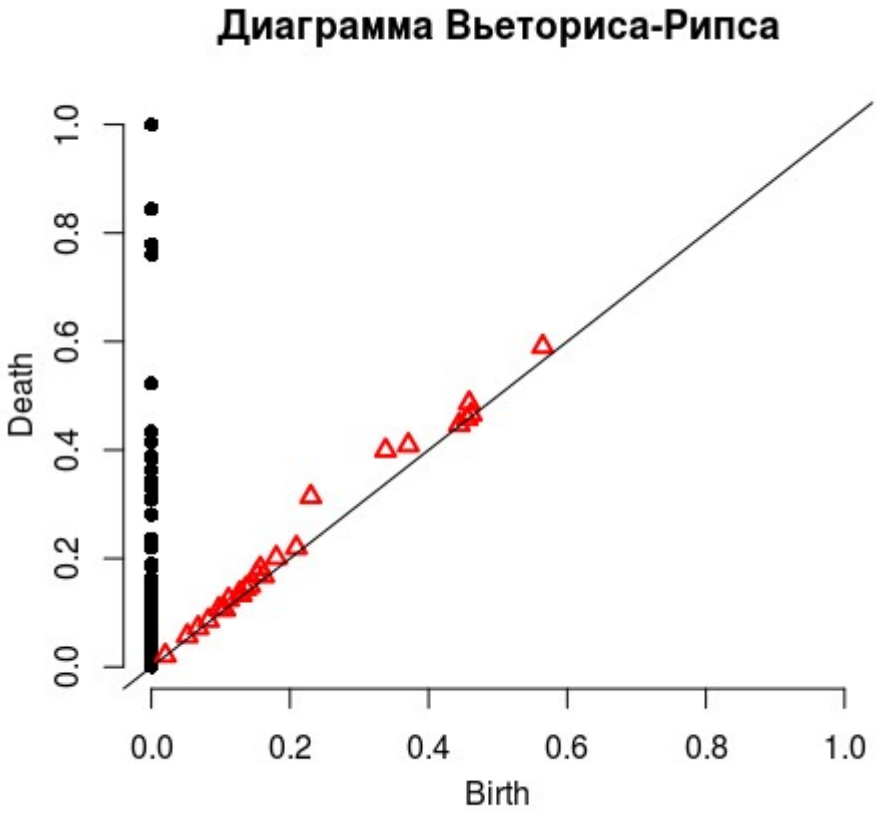

Рисунок 12: Диаграмма работы фильтра Вьеториса-Рипса.

В данной работе из этого пакета используются функции фильтрации Вьеториса-Рипса и построение штрихкода для поиска чисел Бетти, соответствующих различным типам задач, которые были выполнены вычислительным кластером, после чего отражены в анализируемом логе.

На рисунке 12 представлена диаграмма результатов фильтрации данных с помощью фильтра Вьеториса-Рипса, можно заметить, что ярко выделяются два типа задач из рассматриваемых данных, они и представляют устойчивые компоненты связности в данных, которые отражены на приведённой диаграмме разными цветами.

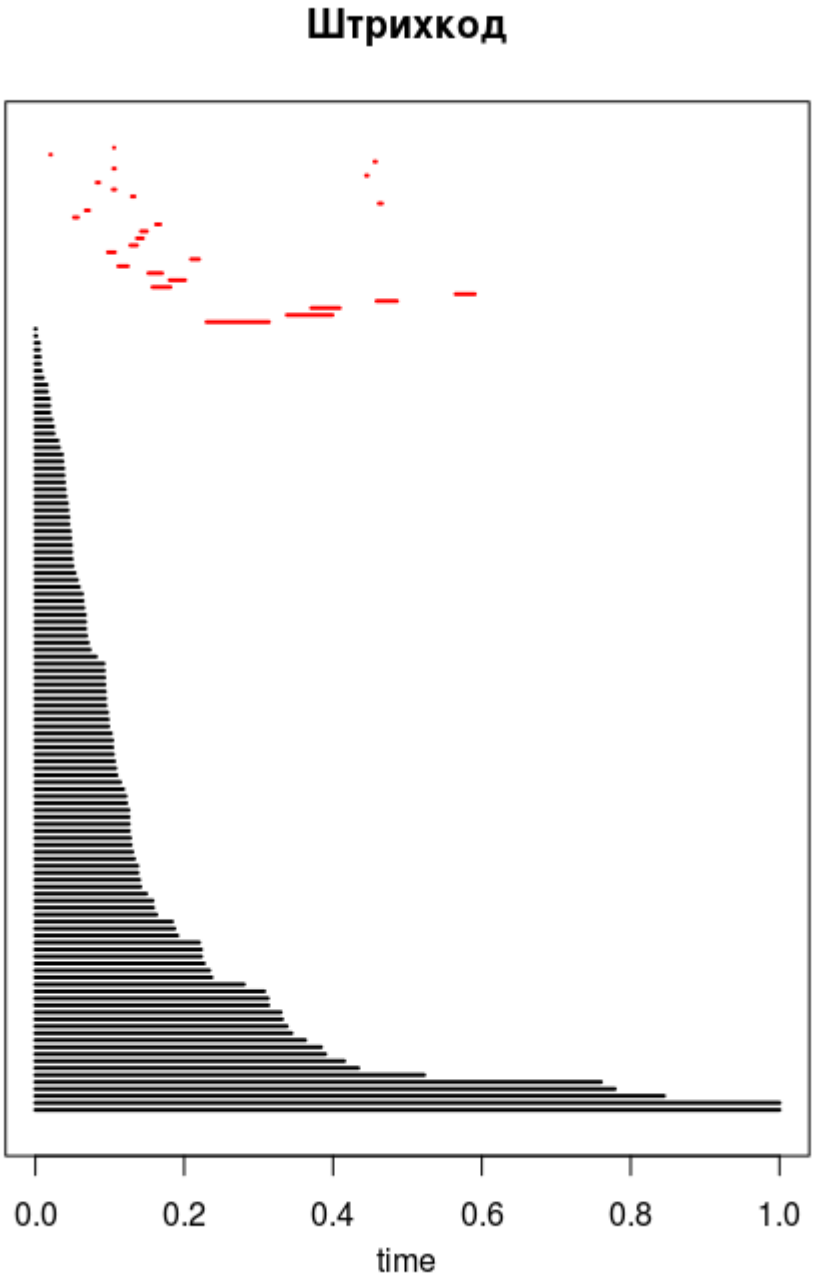

Рисунок 13: Штрихкод отражающий работу фильтра Вьеториса-Рипса.

Помимо этого, разделение с помощью фильтра Вьеториса-Рипса позволяет выделить гомологии существующие в рассматриваемых данных и неизменные на протяжении всего времени применения фильтра. Время жизни таких гомологий как раз и отражает штрихкод.

Представленный на рисунке 13 штрихкод показывает время жизни различных гомологий в рассматриваемых данных в процессе применения фильтра Вьеториса-Рипса, не сложно отметить, что лишь гомологии соответствующие лишь двум числам Бетти являются неизменными и устойчивыми на протяжении всего применения фильтра Вьеториса-Рипса.

3.5 Поиск компонент сильной связности в облаке данных с помощью алгоритма Mapper

В качестве реализации алгоритма Mapper была выбрана библиотека *KeplerMapper* для языка *python*, реализующая алгоритм Mapper на основе модели кластеризации *DBSCAN* из пакета *sklearn.clustering*.

Основная идея алгоритма Mapper заключается в проекции облака точек высокой размерности в пространство меньшей размерности путём «схлопывания» близко расположенных точек в неразделимые кластеры, с последующим выявлением компонент сильной связности в облаке получившихся кластеров.

Каждому из получившихся элементарных кластеров присваивается некоторое значение (в данной работе отображается цветом) на основе свойств, входящих в него элементов.

В качестве входных параметров алгоритм Mapper принимает список измерений изначального облака точек, проекция которых будет осуществляться и процент пересечения гиперкубов.

В данной работе были рассмотрены значения пересечения гиперкубов от 1% до 20% с шагом в 5% для всех сочетаний по 3 измерения из изначальных данных.

В результате выполнения этих операций и перебора сочетаний проекций и степени пересечения гиперкубов были выявлены следующие особенности.

Большинство проекций независимо от степени пересечения гиперкубов выглядели примерно как проекция представленная на рисунке 14. Такой вид проекции означает, что использованные в этой проекции измерения не представляют характеристики способные хорошо разделить данные на категории.

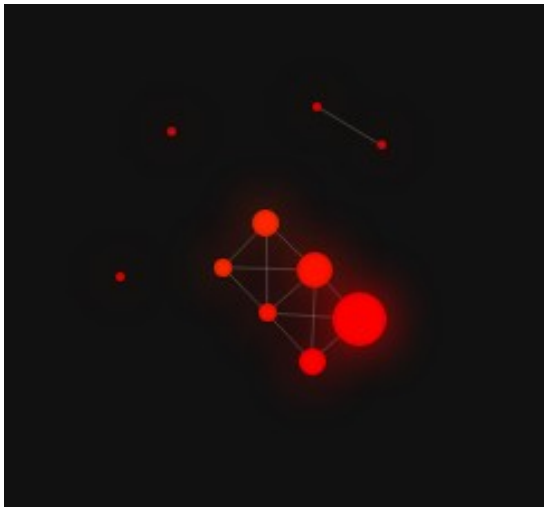

Рисунок 14: Пример "плохой" проекции алгоритма tapper.

Был проведён подробный анализ составленных проекций. Наиболее качественными получились проекции содержащие одновременно характеристики отвечающие за затраченные ресурсы процессора, использование ОЗУ или дисковой памяти и CPI.

На рисунке 15 представлена проекция относительно использования CPU и ОЗУ. Явно видны две компоненты сильной связности отличающиеся цветом (красный и жёлтый).

На рисунке 16 представлена проекция относительно использования CPU и дисковой памяти, как и в предыдущем случае явно видны две компоненты сильной связности отмеченные разными цветами.

На рисунке 17 представлена проекция относительно использования CPU и дисковой памяти, как и в предыдущем случае явно видны две компоненты сильной связности отмеченные различными оттенками красного и жёлтого.

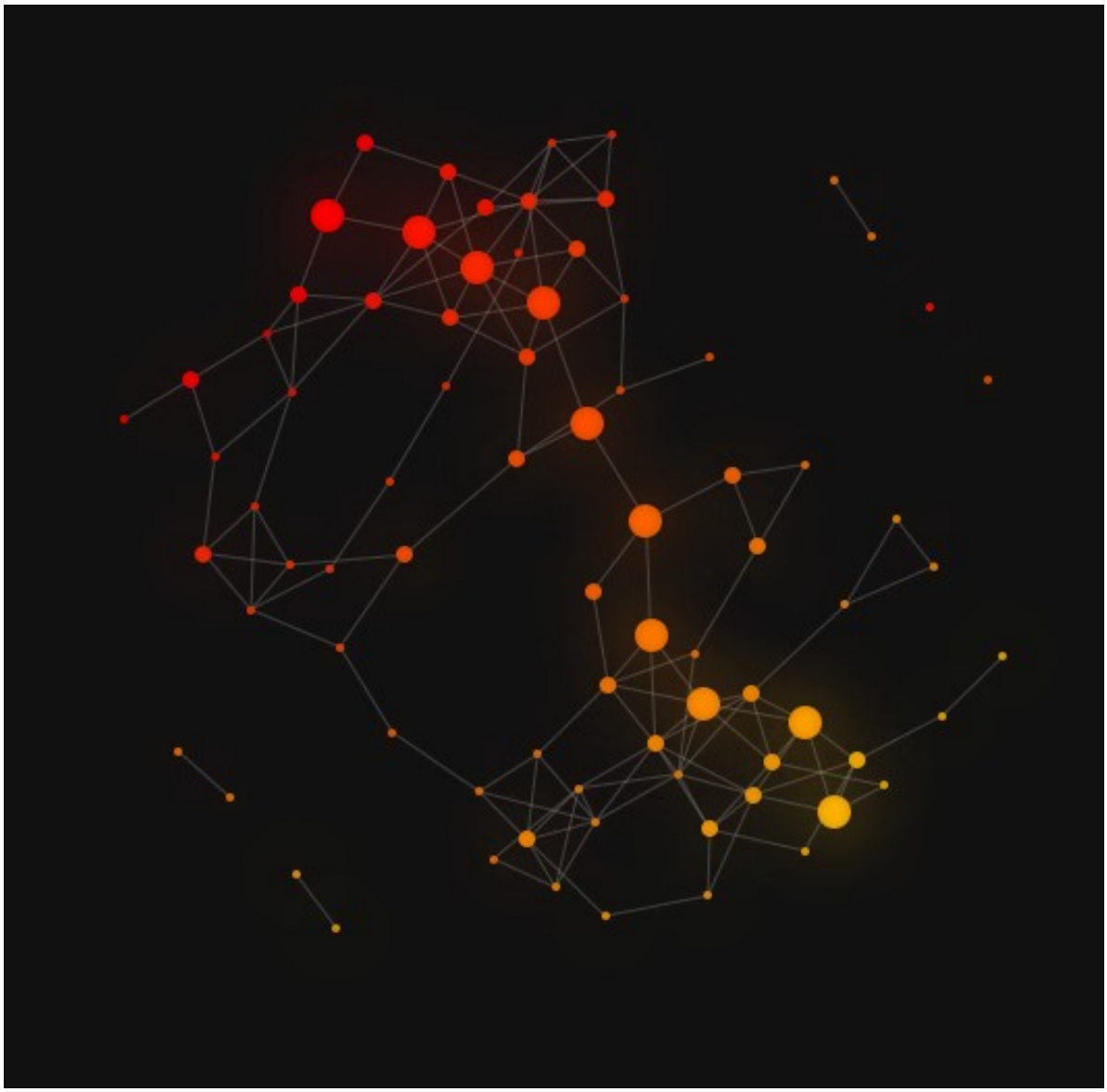

Рисунок 15: Проекция отноительно затраченных ресурсов процессора и ОЗУ.

Как и в случаях кластеризации методом k-средних и устойчивых гомологий с помощью алгоритма Mapper удалось выделить и визуализировать два раздельных типа задач встречающихся в логе кластера Google.

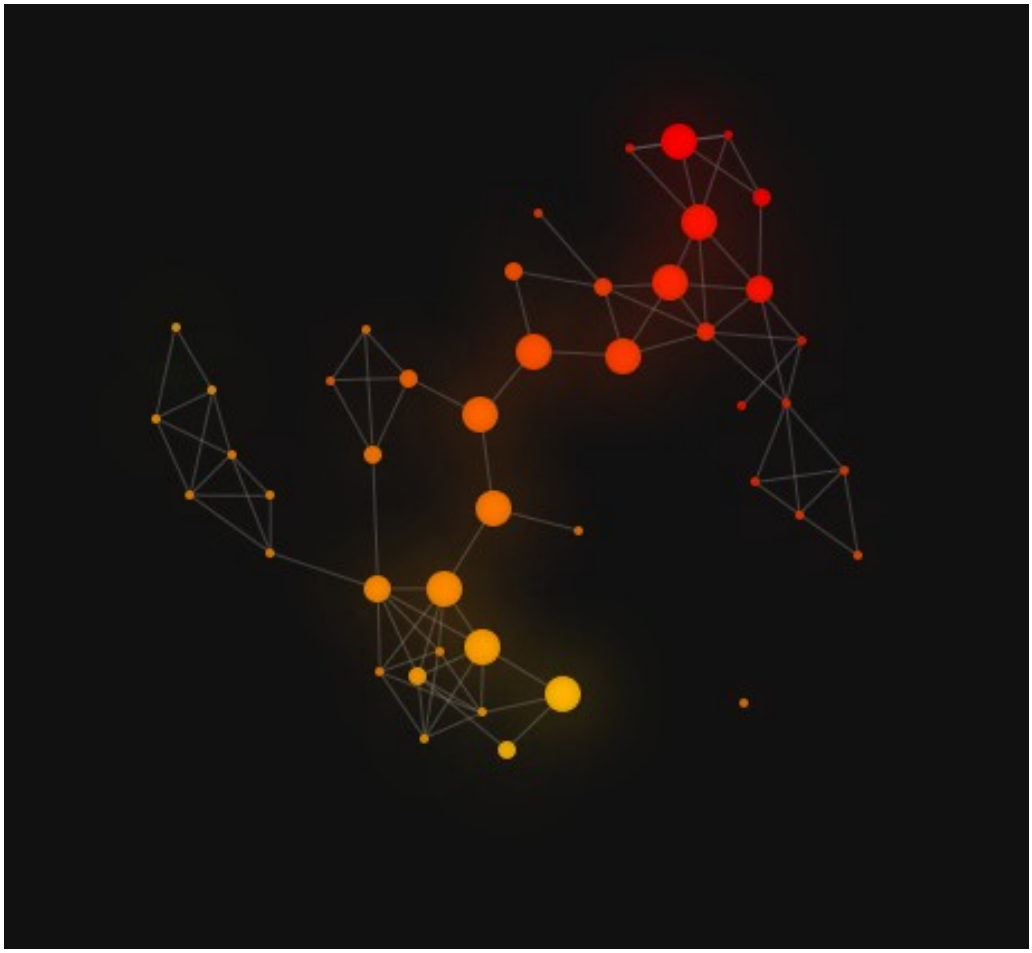

Рисунок 16: Проекция относительно использования CPU и дисковой памяти.

Рисунок 17: Проекция относительно использования CPU и показателя CPI.

Выводы по главе

В результате подробного анализа данных, входящих в каждый из выделенных рассматриваемыми алгоритмами кластеров (компонент сильной связности) были выявлены особенности присущие задачам, размещённым в этих кластерах (компонентах связности).

В трёх из рассмотренных четырёх случаев с помощью алгоритмов анализа данных были выявлены два типа задач, встречающихся в логе кластера Google. После рассмотрения типов задач, выделенных каждым из «успешных» алгоритмов был сделать вывод о том, что типы задач, выделенные каждым из алгоритмов, очень похожи между собой и представляют собой два класса задач:

1. задачи работы с данными,
2. вычислительные задачи.

Первый из этих двух типов характеризуется интенсивной нагрузкой памяти, как дисковой, так и ОЗУ, более высоким показателем CPI. Второй класс характеризуется более высокой нагрузкой на CPU.

Помимо этого, были произведены замеры времени работы каждого из предложенных алгоритмов и затребованная ими память. Существенно быстрее и намного менее требовательным оказался алгоритм кластеризации методом k-means. По этим показателям он уступил только алгоритму смеси гауссиан с процессом Дирихле, однако результаты последнего оказались некачественными.

Алгоритм поиска устойчивых гомологий и Mapper показали себя как мощные инструменты анализа данных с чрезвычайно высокой степенью корректности результатов, однако в такой же высокой степени требовательные к ресурсам и времени.

Полученные результаты позволяют сделать вывод о том, что топологические методы анализа данных прекрасно подходят для поиска скрытых особенностей в данных и верификации работы традиционных алгоритмов анализа данных, однако

в силу требовательности ко времени и ресурсам их использование для встраивания в системы балансировки нагрузки в вычислительной сети не рационально.

Таким образом топологические методы анализа данных применимы в задачах балансировки нагрузки в вычислительной сети, однако не в качестве части работающего балансировщика нагрузки, а лишь в качестве инструмента анализа и верификации его работы и анализа логов входящих задач/трафика с целью выявления скрытых особенностей.

Заключение

Целью данной работы была проверка гипотезы о том, что применение топологических методов анализа данных позволит выбирать более эффективные стратегии балансировки нагрузки в вычислительной сети, за счёт выявления дополнительных особенностей, не выявляемых традиционными методами анализа данных.

Был проанализирован набор данных описывающих месяц работы вычислительных кластеров Google.

Были реализованы и применены несколько методик выявления особенностей задач, выполняемых вычислительным кластером Google.

Результаты работы топологических и традиционных методов анализа данных были рассмотрены и проанализированы. На основе проведённого анализа были сделаны выводы о применимости топологического анализа данных и конкретных аспектах его применения в задаче балансировки нагрузки в вычислительной сети.

Список литературы

1. John Wilkes, Charles Reiss, ClusterData2011_2 traces,

2. ГОСТ 22402-88 Телеобработка данных и вычислительные сети. Термины и определения., 2010

3. K. A. Nuaimi, N. Mohamed, M. A. Nuaimi, J. Al-Jaroodi, A Survey of Load Balancing in Cloud Computing: Challanges and Algorithms., 2012

4. R. G. Rajan, V. Jeyakrishnan, A Survey on Load Balancing in Cloud Computing Environments, 2013 International Journal of Advanced Research in Computer and Communication Engineering, Vol. 2, Iss. 12

5. D. Kashyap, J. Viradiya, A Survey of Various Load Balancing Algorithms in Cloud Computing., 2014 International Journal of Scientific & Technology Research, Vol. 3, Iss. 11

6. M. Lesnick, M. Wright, Interactive Visualisation of 2-D Persistence Modules, 1512.00180v1

7. Peter Bubenik, Statistical Topological Data Analysis using Persistence Landscapes., 2015 Journal of Machine Learning Research

8. R. Ghrist, Barcodes: The Persistent Topology of Data., 2008 Bulletin of The American Mathematical Society

9. G. Carlsson, R. Jardine, D. Feichtner-Kozlov, D. Morozov., Topological Data Analysis and Machine Learning Theory., 2012

10. Jesse Johnson, Topological Graph Clustering With Thin Position., arXiv:1206.0771

11. M. Scharlemann, A. Thompson, Thin Position for 3-Mainfolds., 1994 Contemporary Mathematics

12. M. Belkin, K. Sinha, Polynomial Learning of Distribution Families., 2010 Proceeding of Annual IEEE Symposium on Foundations of Computer Science

13. T. Hastie, E. Tibshirani, J. Friedman, The Elements of Statistical Learning, 2009, rev. 2013

14. G. Lerman, M. B. McCoy, J. A. Tropp, T. Zhang, Robust Computation of Linear Models by Convex Relaxaion, 2012 1202.4044v2

15.: H. Edelsbrunner, J. Harer, Persistent Homology — a Survey, 2008

16. V. de Silva, R. Ghrist, Coverage in sensor networks via persistent homology., 2007 Algebraic & Geometric Topology 7

17. M. Nicolau, A. J. Levine, G. Carlsson, Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival., 2011 PNAS

18. M. A. R. Dantas, A. R. Pinto, A load balancing approach based on a genetic machine learning algorithm, 2005 19th International Symposium on High Performance Computing Systems and Applications (HPCS'05)

19. R. F. de Mello, J. A. A. Filho, L. J. Senger, L. T. Yang, RouteGA: A Grid Load Balancing Algorithm with Genetic Support, 2007 21st International Conference on Advanced Networking and Applications(AINA'07)

20. R. F. de Mello, L. J. Senger, and L. T. Yang, A routing loadbalancing policy for grid computing environments., 2006 In TheIEEE 20th International Conference on Advanced Informa-tion Networking and Applications (AINA 2006)

21. J. Li, X. Ma, K. Singh, M. Schulz, B. R. de Supinski, S. A. McKee, Machine Learning Based Online Performance Prediction for Runtime Parallelization and Task Scheduling, 2009

22. Y. Chen, A. S. Ganapathi, R. Griffith, R. H. Katz, Analysis and Lessons from a Publicly Available Google Cluster Trace, 2010 Technical Report No. UCB/EECS-2010-95

23. S. Lloyd , Least squares quantization in PCM, 1982 IEEE Transactions on Information Theory

24. Y. W. Teh, M. I. Jordan, M. J. Beal, D. M. Blei, Hierarchical Dirichlet Processes, 2006 Journal of the American Statistical Association, Vol. 101, No. 476

25. , Gaussian mixture models, http://scikit-learn.org/stable/modules/mixture.html

26. A. P. Dempster, N. M. Laird, D. B. Rubin, aximum likelihood from incomplete data via the EM algorithm., 1977 Journal of the Royal Statistical Society. Series B (Methodological)

27. R. Hennigan, A Fast Simplicial Complex Construction for Computing the Persistent Homology of Very Large, High Dimentional Data Sets, 2014

28. G. Singh , F. Mémoli, G. Carlsson, Topological Methods for the Analysis of High DimensionalData Sets and 3D Object Recognition, 2007 Eurographics Symposium on Point-Based Graphics

29. P. J. Rousseeuw, Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis, 1987 Computational an Applied Mathematics

30. Scikit-learn Community, Документация Sklearn, http://scikit-learn.org/stable/modules/mixture.html

31. B. T. Fasy, J. Kim, F. Lecci, C. Maria, Introduction to the R package TDA

Приложение 1. Результаты кластеризации и силуэты для метода k-means.

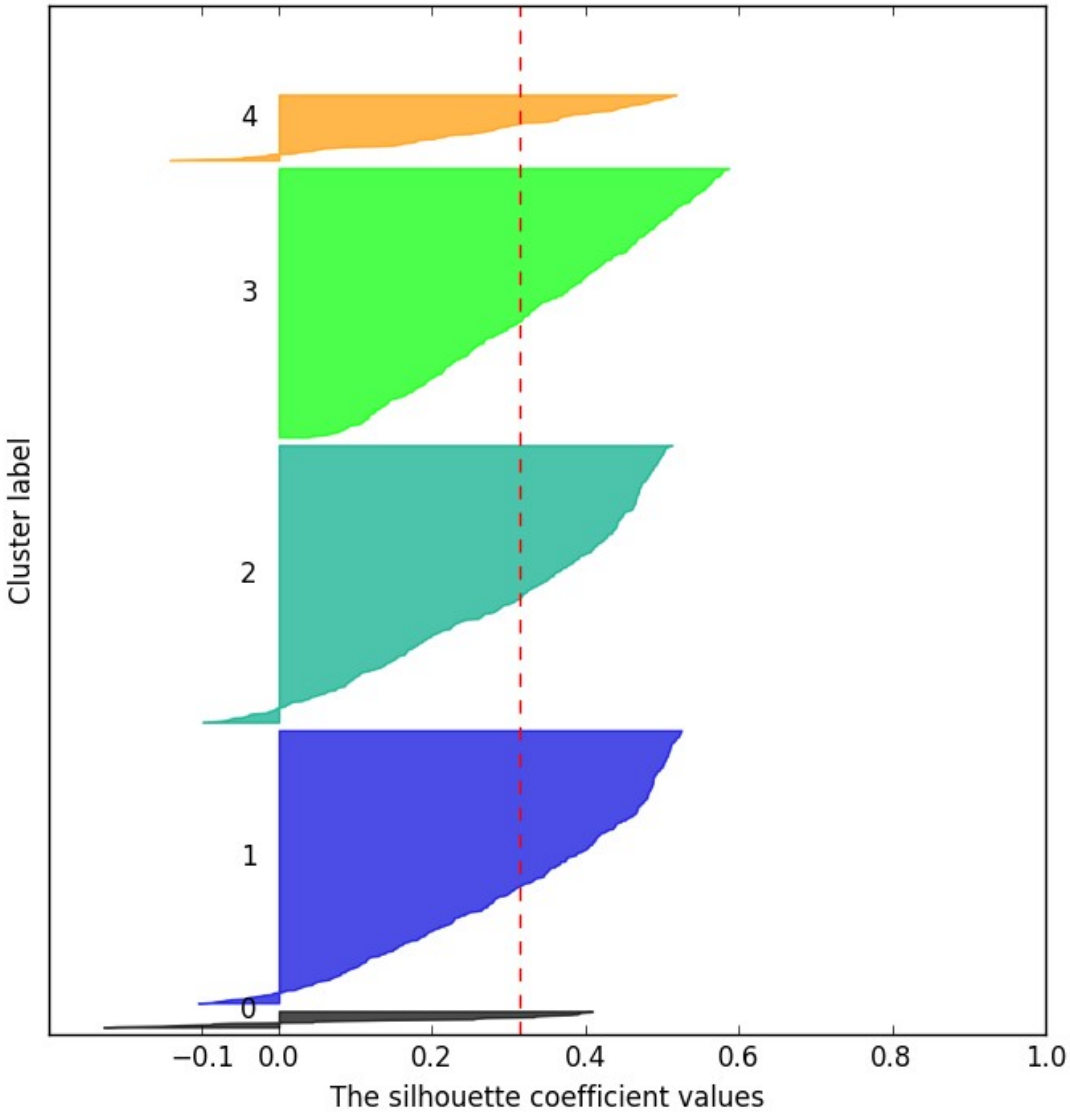

Рисунок 18: Силуэты 5 кластеров.

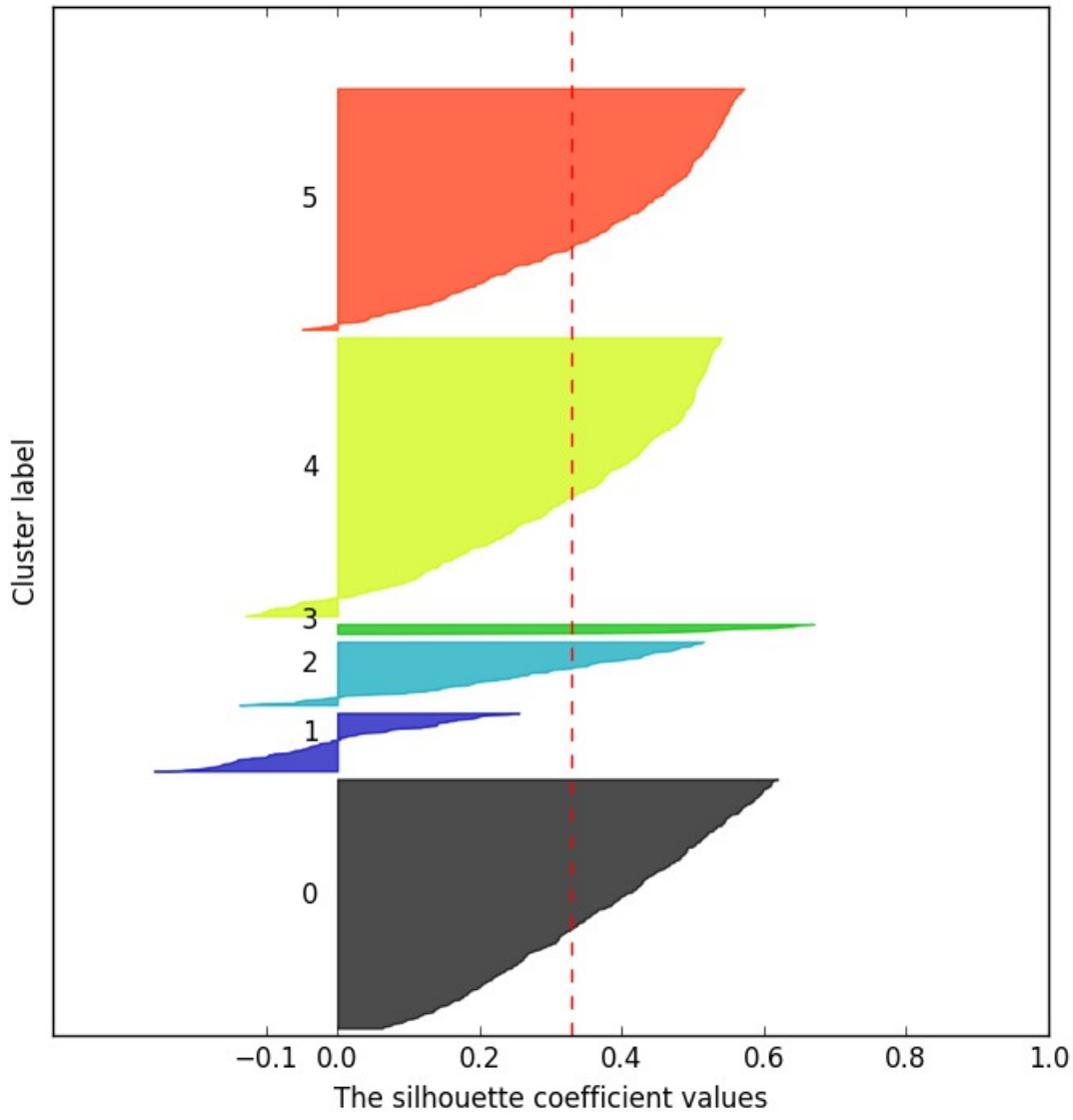

Рисунок 19: Силуэты для 6 кластеров.

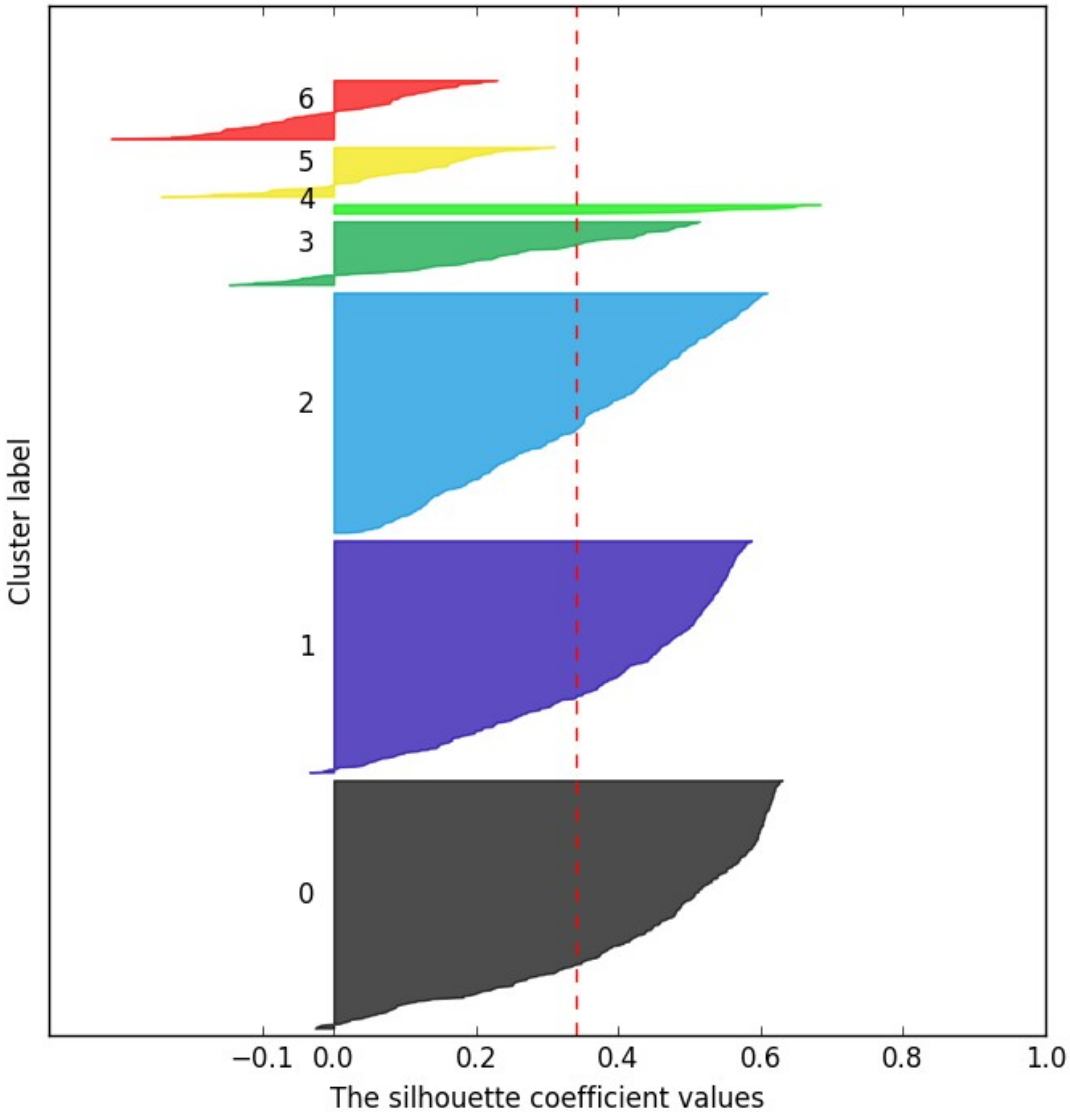

Рисунок 20: Силуэты 7 кластеров.

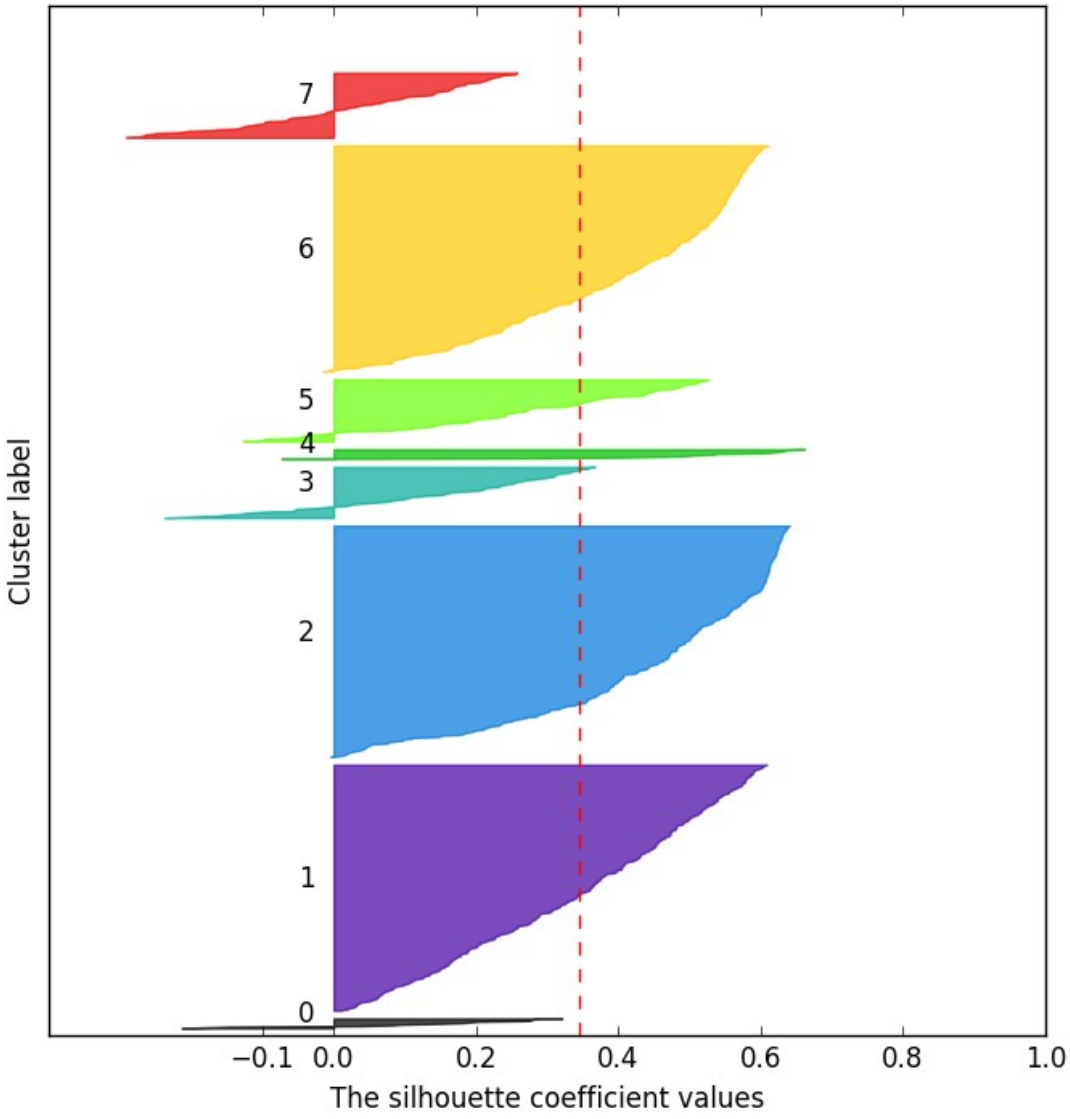

Рисунок 21: Силуэты 8 кластеров.

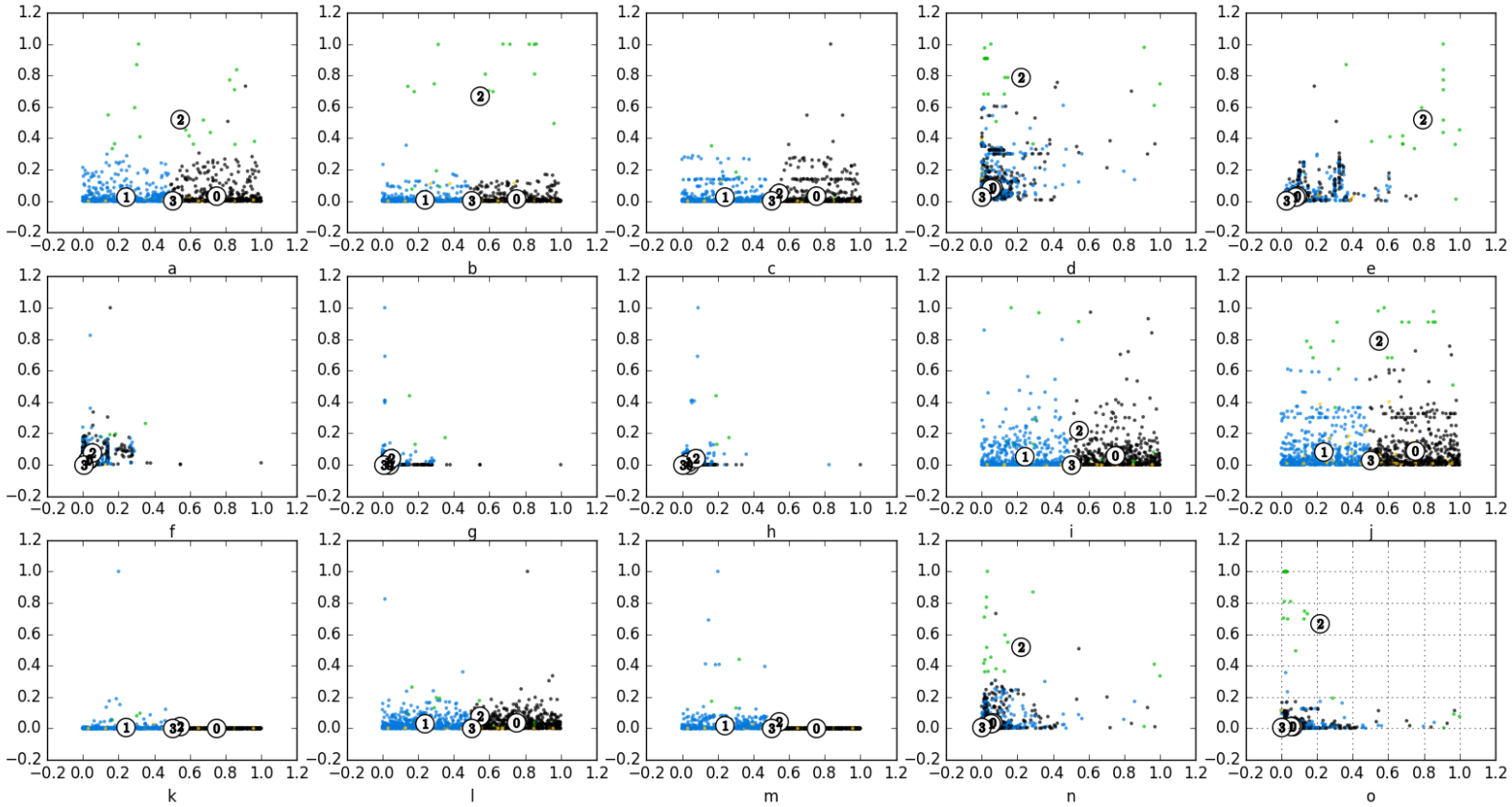

Рисунок 22: Разбиение облака данных на 4 кластера.

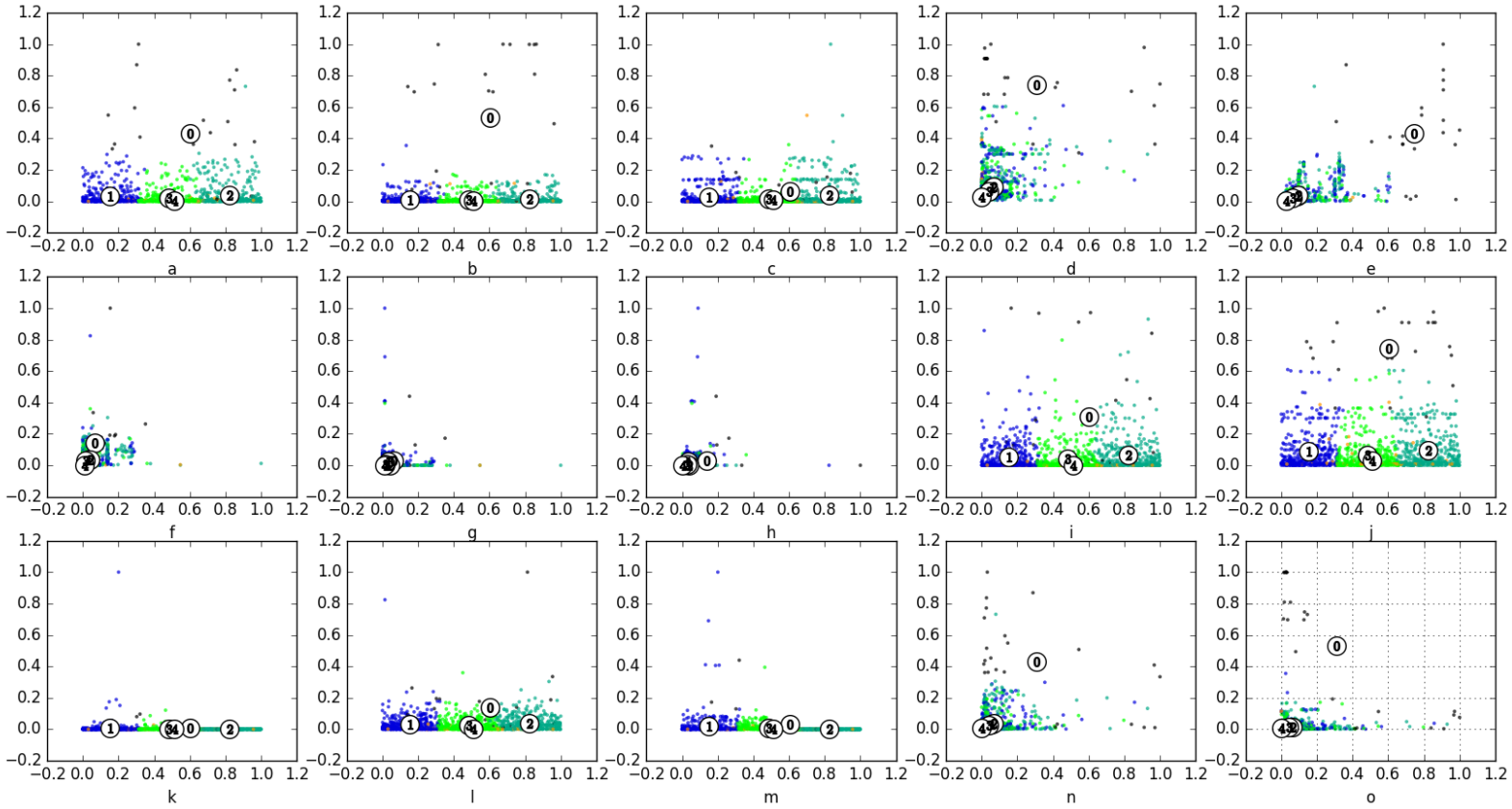

Рисунок 23: Разбиение облака данных на 5 кластеров.

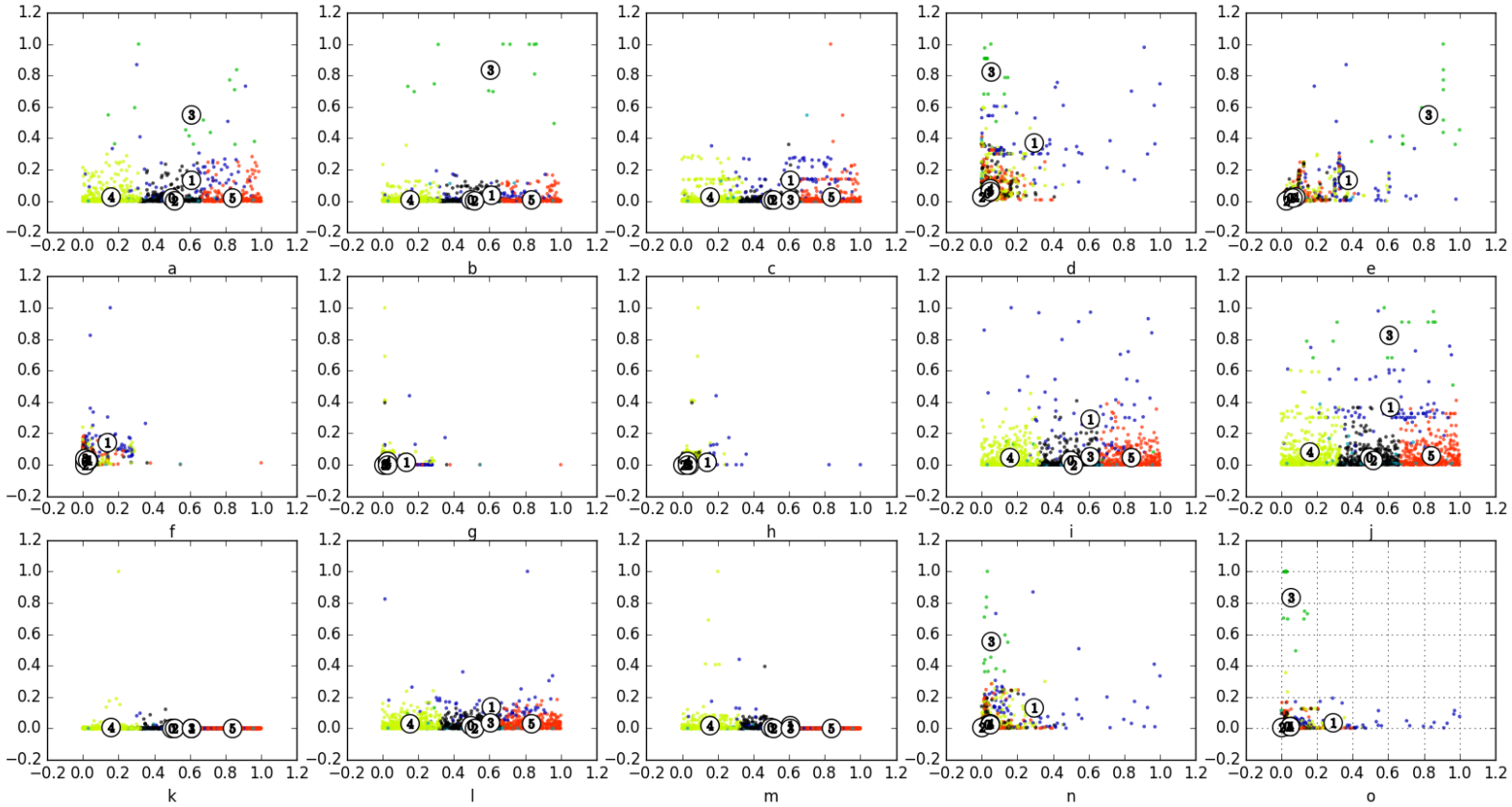

Рисунок 24: Разбиение облака данных на 6 кластеров.

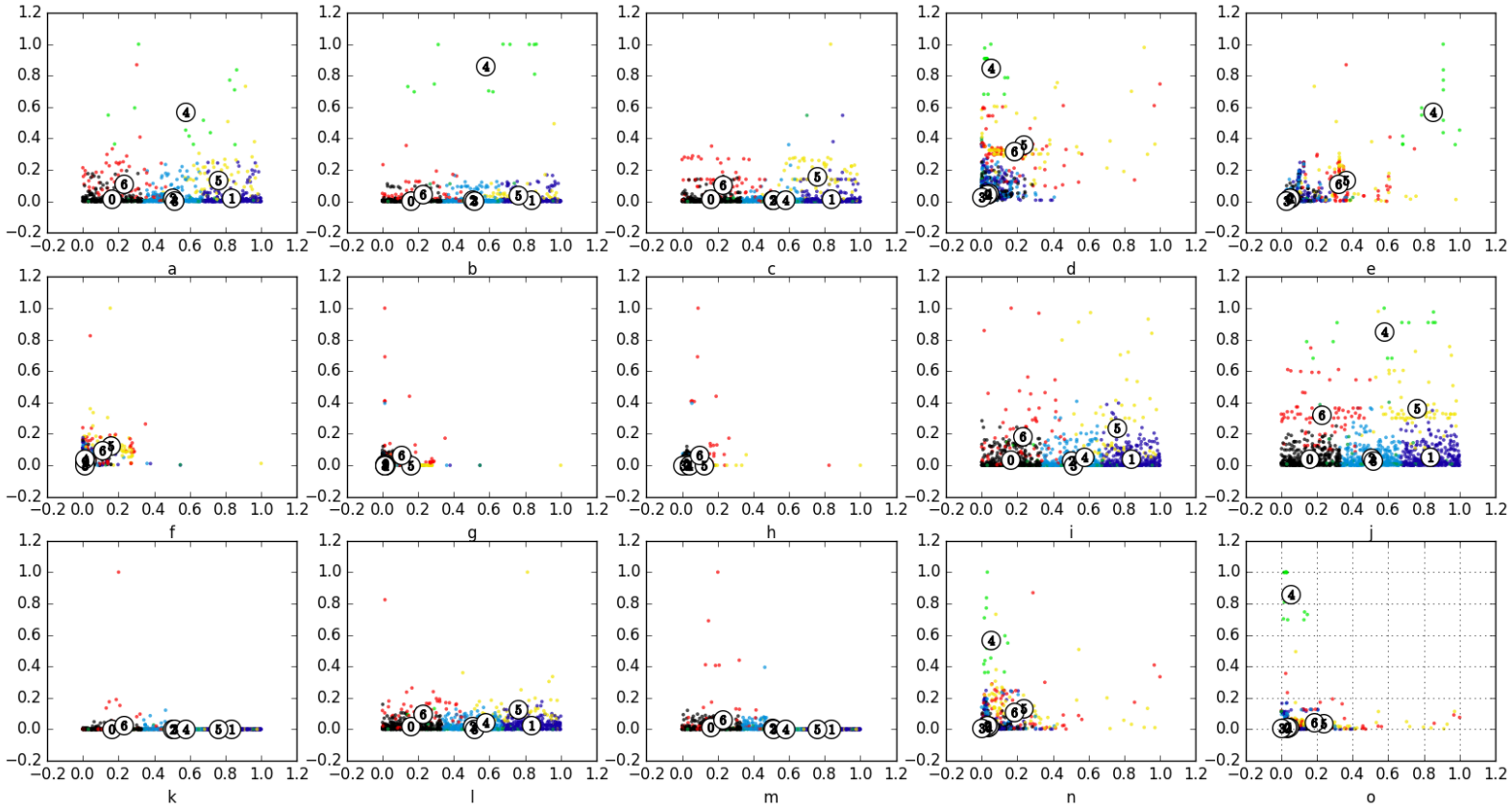

Рисунок 25: Разбиение облака данных на 7 кластеров.

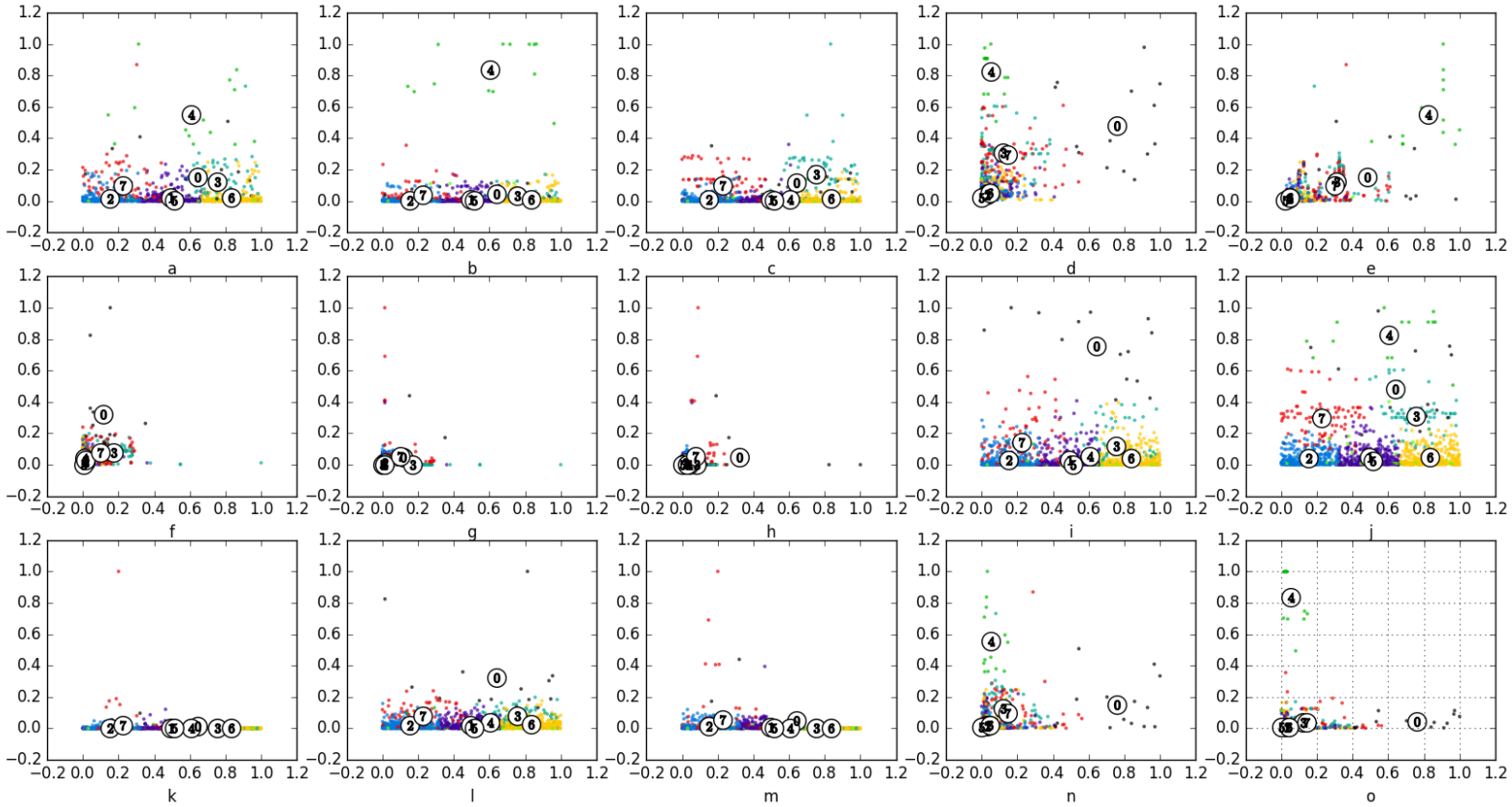

Рисунок 26: Разбиение облака данных на 8 кластеров.

Приложение 2. Результаты кластеризации и силуэты для метода DPGMM.

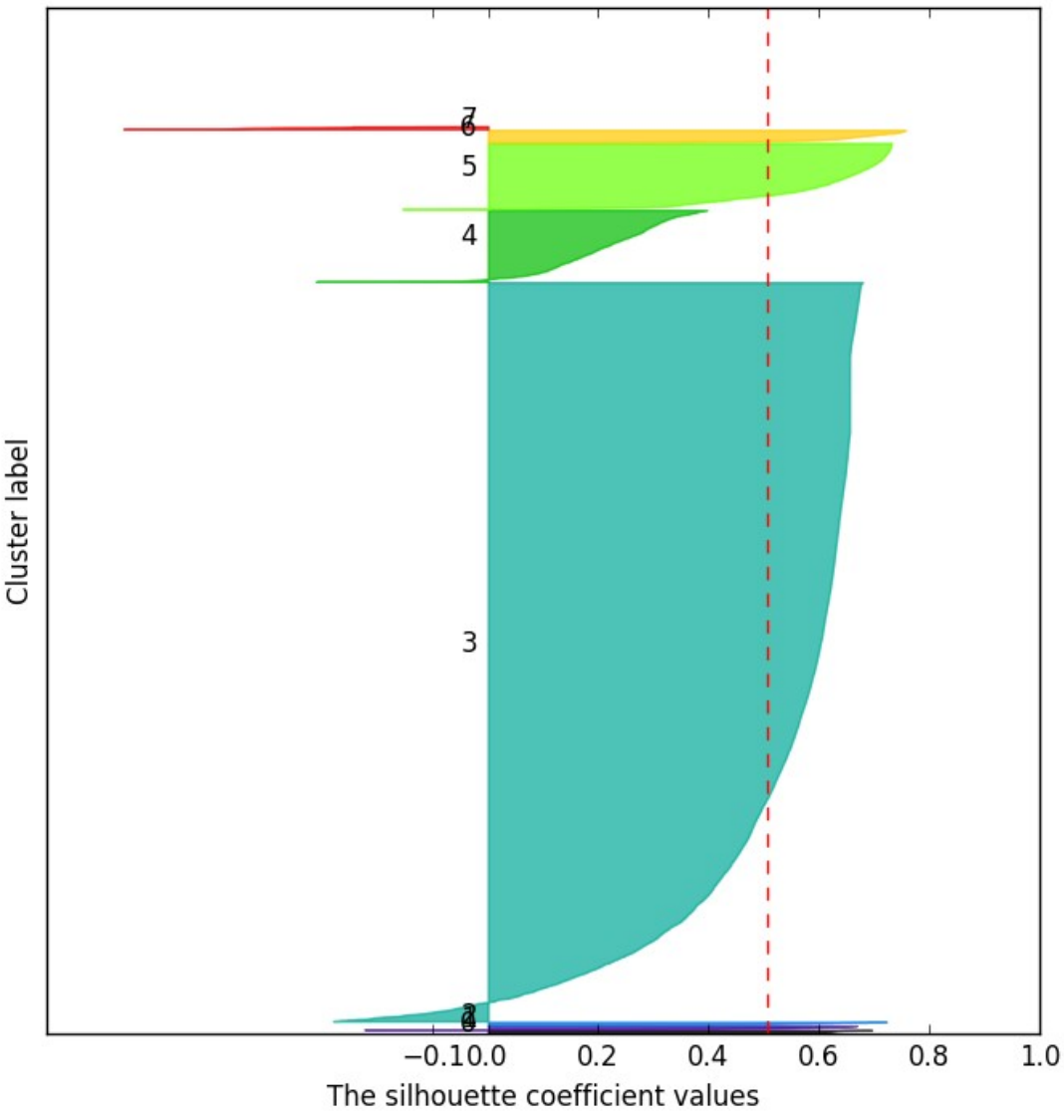

Рисунок 27: Силуэты 8 кластеров для $\alpha = 10$.

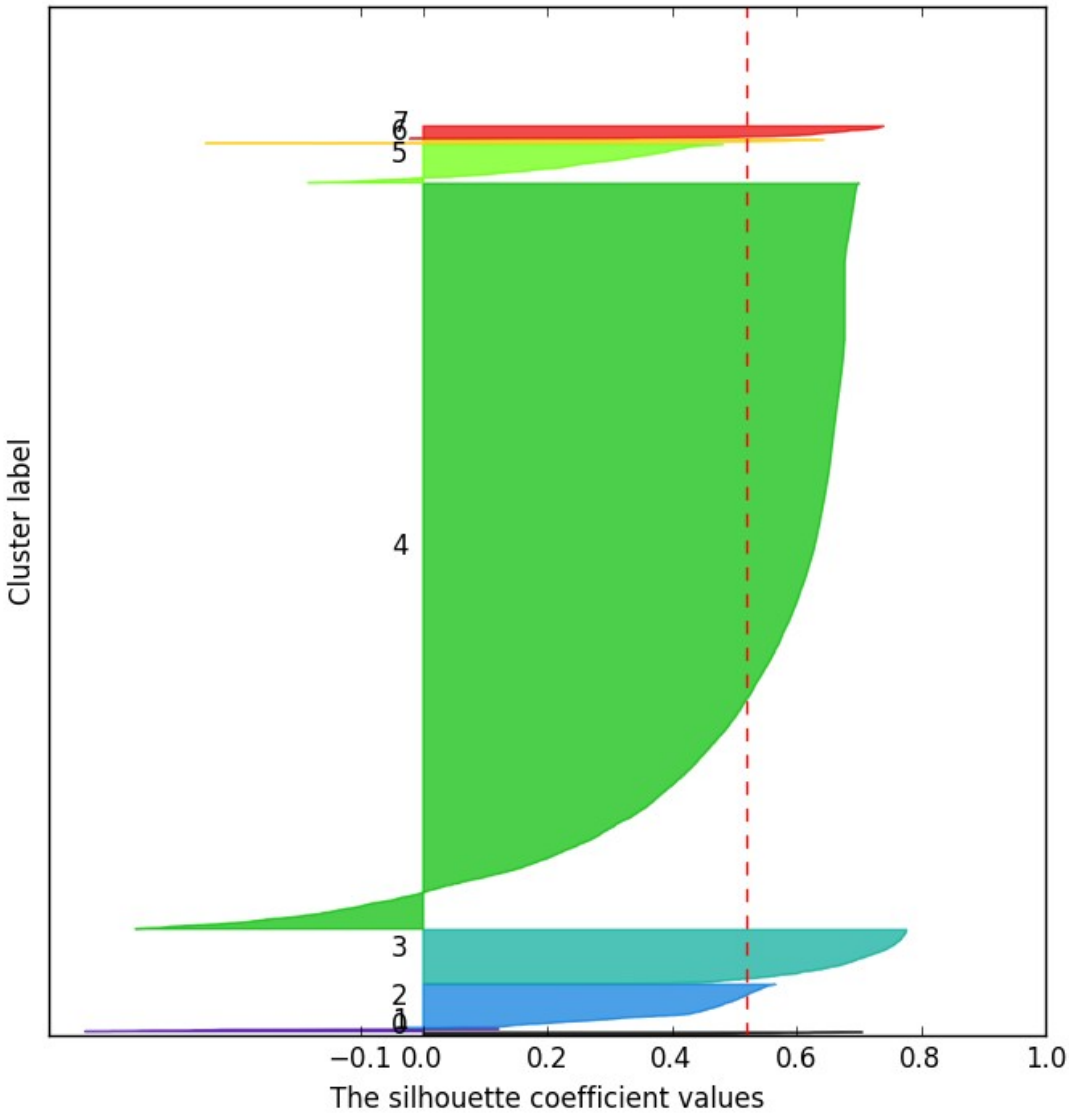

Рисунок 28: Силуэты 8 кластеров для $\alpha = 100$.

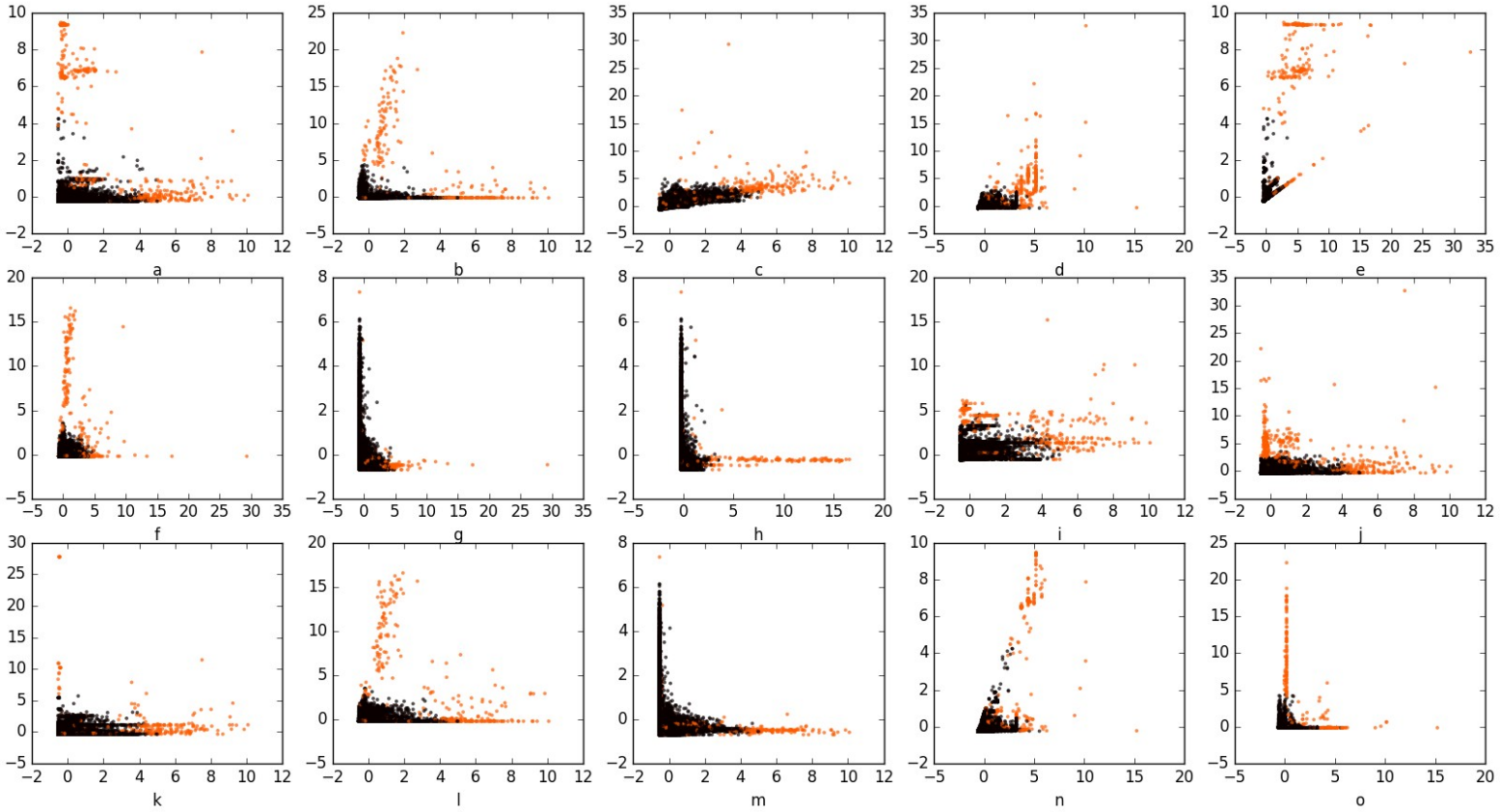

Рисунок 29: Разбиение облака данных на 2 кластера с помощью DPGMM.

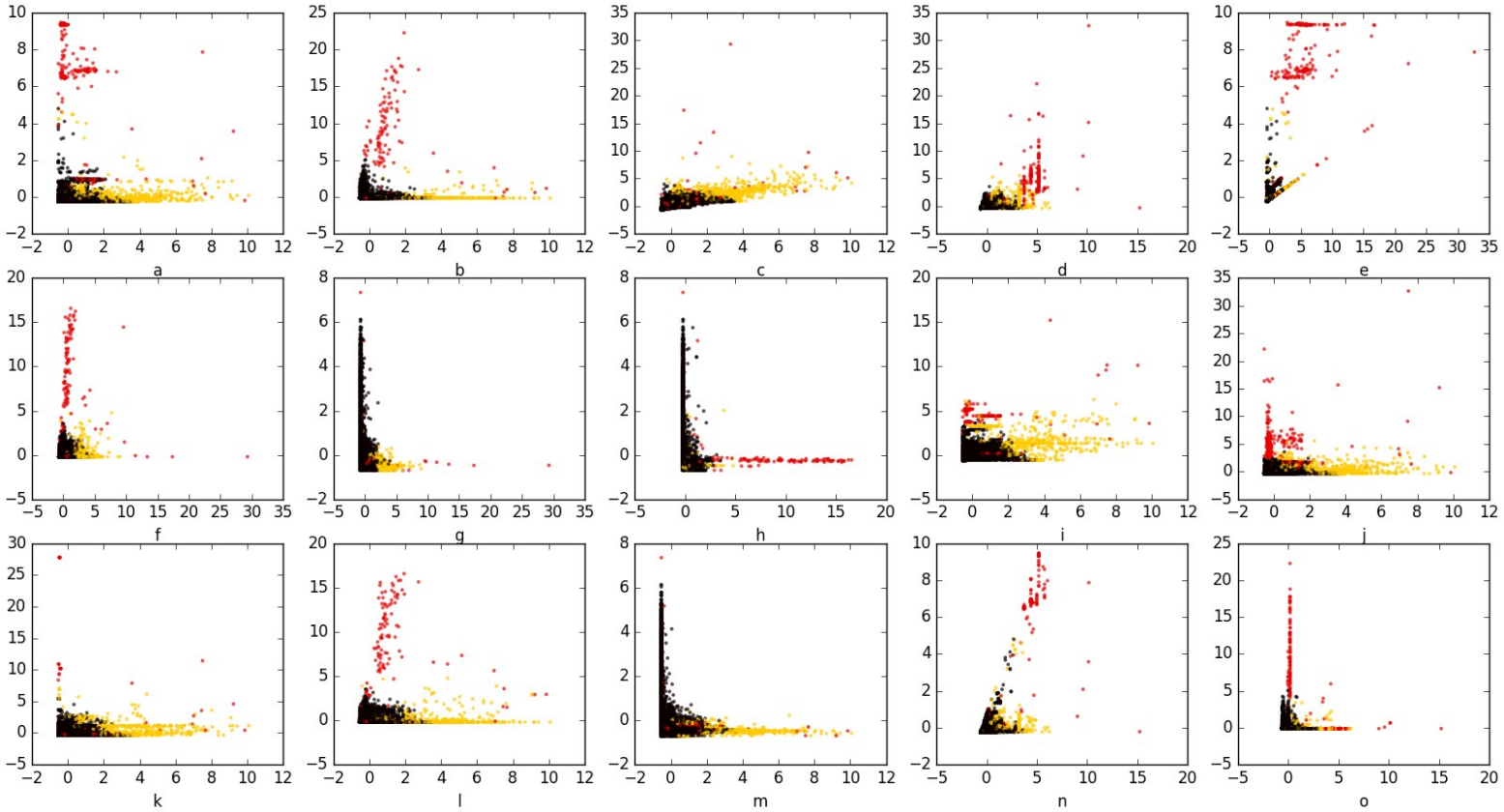

Рисунок 30: Разбиение облака данных на 3 кластера с помощью DPGMM.

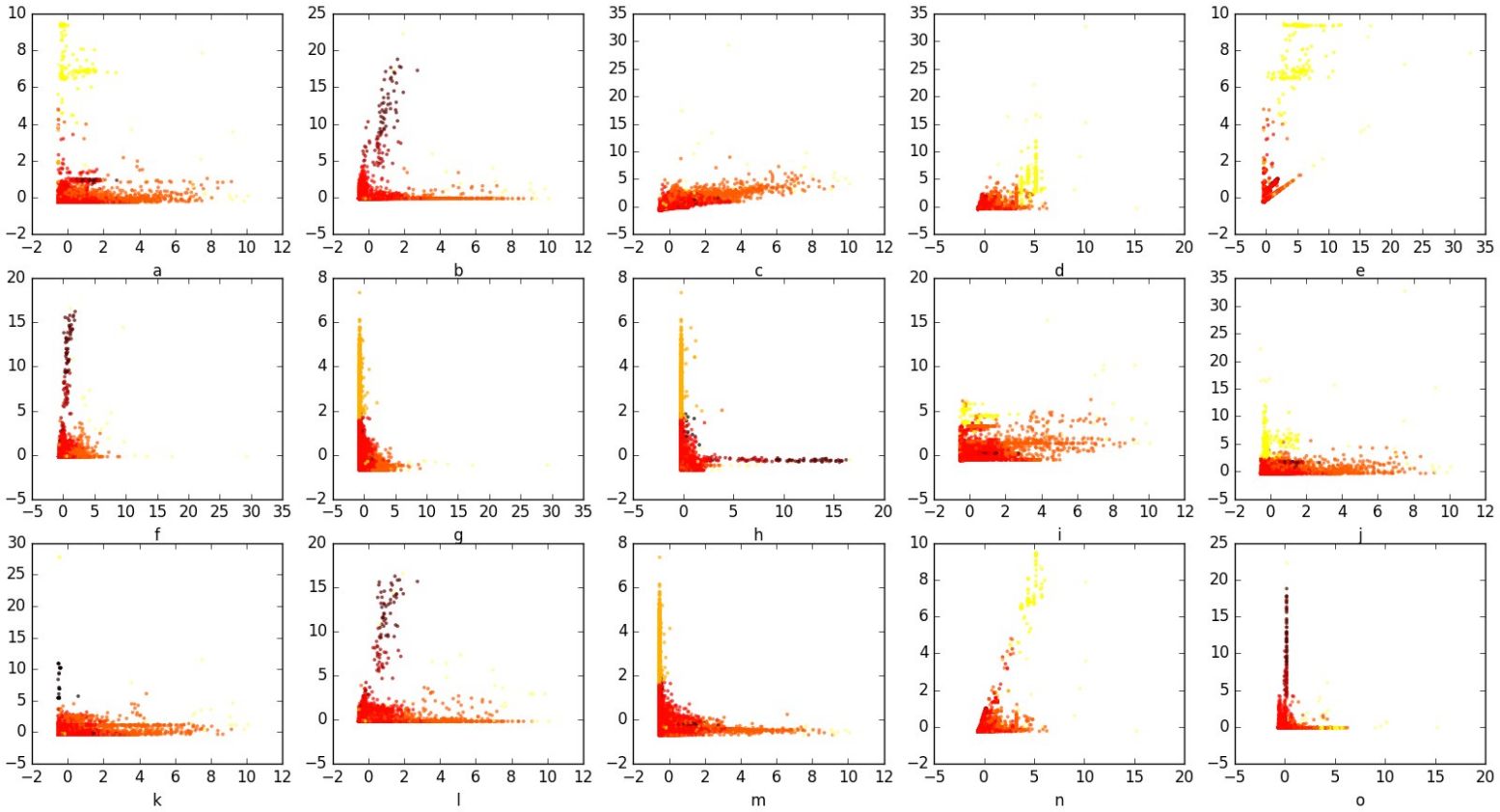

Рисунок 31: Разбиение облака данных на 8 кластеров с помощью DPGMM.

Приложение 3. Выдержки из программного кода.

Построение выборок данных.

```
# -*- coding:utf-8 -*-

import pandas as pd
from os import listdir
from os.path import isfile, join

datapath = './task_usage'
columns_to_drop = [0, 1, 2, 3, 4, 17, 18, 19]

# Получение списка файлов из директории с данными
filelist = [join(datapath,filename) for filename in listdir(datapath) if isfile(join(datapath,filename))]

for i in range(0,10):
    current_file = pd.read_csv(filelist[0], header=None)
    # Переменная для хранения выборки содержащей 0.1% от всех данных
    df_01p = current_file.sample(frac=0.001, random_state=312)
    for filename in filelist[1:]:
        current_file = pd.read_csv(filename, header=None)
        df_01p = pd.concat([df_01p, current_file.sample(frac=0.001, random_state=312)])

    df_01p = df_01p.drop(df_01p.columns[columns_to_drop], axis=1)
```

```
    df_01p = df_01p.fillna(0)
    df_01p.to_csv('sample_01p_num0%i.csv' % i )

print "0.1% data preparation finished"

for i in range(0,10):
    current_file = pd.read_csv(filelist[0], header=None)
    # Переменная для хранения выборки содержащей 1% от всех данных
    df_01p = current_file.sample(frac=0.01, random_state=312)
    for filename in filelist[1:]:
        current_file = pd.read_csv(filename, header=None)
        df_01p = pd.concat([df_01p, current_file.sample(frac=0.001, random_state=312)])

    df_01p = df_01p.drop(df_01p.columns[columns_to_drop], axis=1)
    df_01p = df_01p.fillna(0)
    df_01p.to_csv('sample_1p_num0%i.csv' % i )

print "1% data preparation finished"
```

Преобразование данных

```
# -*- coding:utf-8 -*-

import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np
```

```
print ">>>Starting Data Load Process"
data = pd.read_csv(argv[1], index_col=0).as_matrix()
print "<<<Data Loaded"

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
print ">>>Starting Data Scaling Process"
scaled_data = scaler.fit_transform(data)
print "<<<Data Scaled"

print ">>>Starting Data Labling Process"
df = pd.DataFrame(scaled_data)

df = df.rename(index=str, columns={
    0:'Mean CPU usage rate',               # 0
    1:'Canonical memory usage',            # drop
    2:'Assigned memory usage',             # 1
    3:'Unmapped page cache memory usage',  # 2
    4:'Total page cache memory usage',     # 3
    5:'Maximum memory usage',              # drop
    6:'Mean disk I/O time',                # 4
    7:'Mean local disk space used',        # 5
    8:'Maximum CPU usage',                 # 6
    9:'Maximum disk I/O time',             # 7
```

```
    10:'Cycles per instruction (CPI)',      # 8
    11:'Memory access per instruction (MAI)',# drop
})

print "<<<Data Labeled"
print ">>>Dropping unnecessary columns"
df = df.drop(df.columns[[11,1,5]], axis=1)
print "<<<Columns Dropped"

print ">>>Starting Data Write Process"
df.to_csv(argv[2])
print "<<<Write Finished"
```

Кластеризация k-means, подбор параметров и поиск силуэтов

```
 -*- coding:utf-8 -*-

import pandas as pd
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

data = pd.read_csv('scaled_data/small_dist_sample_scaled.csv', index_col=0)
```

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

k_means_cluster_num = range(2,9)
X = data.as_matrix()
for n_clusters in k_means_cluster_num:

    print ">>>Clustering"
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X)
    print ">>>Silhoutte"
    silhouette_avg = silhouette_score(X, cluster_labels, sample_size=3000)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)
    sample_silhouette_values = silhouette_samples(X, cluster_labels)
```

Кластеризация с помощью DPGMM

```
# -*- coding:utf-8 -*-

import pandas as pd
import matplotlib
matplotlib.use('Agg')
from scipy import linalg
import matplotlib.pyplot as plt
import matplotlib.cm as cm
```

```
import numpy as np

data = pd.read_csv(argv[1], index_col=0)

from sklearn.mixture import DPGMM
from sklearn.preprocessing import scale
from sklearn.metrics import silhouette_samples, silhouette_score

X = data.as_matrix()
X = scale(X)
alpha_values = [0.005, 0.01, 0.1, 1, 10]
for num, alpha in enumerate(alpha_values):
    print ">>> For Alpha", alpha
    dpgmm = DPGMM(n_components=9, covariance_type='spherical', alpha=alpha)
    print ">>>Clustering"
    dpgmm.fit(X)
    cluster_labels = dpgmm.predict(X)

    print ">>>Silhoutte"
    silhouette_avg = silhouette_score(X, cluster_labels, sample_size=3000)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)
    sample_silhouette_values = silhouette_samples(X, cluster_labels)
```

Отображение кластеров.

```
fig, ((ax11, ax12, ax13, ax14, ax15),
```

```
        (ax21, ax22, ax23, ax24, ax25),
        (ax31, ax32, ax33, ax34, ax35)) = plt.subplots(3, 5, figsize=(20,10))
    feature_pairs = [
        [ax11,0,3,'a'],
        [ax12,0,4,'b'],
        [ax13,0,6,'c'],
        [ax14,1,2,'d'],
        [ax15,2,3,'e'],
        [ax21,6,7,'f'],
        [ax22,6,8,'g'],
        [ax23,7,8,'h'],
        [ax24,0,1,'i'],
        [ax25,0,2,'j'],
        [ax31,0,5,'k'],
        [ax32,0,7,'l'],
        [ax33,0,8,'m'],
        [ax34,1,3,'n'],
        [ax35,1,4,'o']]

    print ">>>Plotting"
    print np.unique(cluster_labels)
    n_clusters = np.unique(cluster_labels).shape[0]
    for feature_pair in feature_pairs:
        colors = cm.hot(cluster_labels.astype(float) / (np.unique(cluster_labels).max()+1))
        feature_pair[0].scatter(X[:, feature_pair[1]], X[:, feature_pair[2]], marker='.', s=30,
lw=0, alpha=0.7,
```

```
                c=colors)
        feature_pair[0].set_xlabel(feature_pair[3])
    plt.savefig('results/dpgmm/dist_data/clusters_%i.png' % num)
```

Отображение силуэтов.

```
fig.set_figheight(8)
    fig.set_figwidth(8)

    ax = plt.subplot(111)
    y_lower = 10
    for i in range(n_clusters):
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.spectral(float(i) / n_clusters)
        ax.fill_betweenx(np.arange(y_lower, y_upper),
                         0, ith_cluster_silhouette_values,
                         facecolor=color, edgecolor=color, alpha=0.7)

        ax.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
```

```
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax.set_xlabel("The silhouette coefficient values")
    ax.set_ylabel("Cluster label")

    # The vertical line for average silhoutte score of all the values
    ax.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax.set_yticks([])  # Clear the yaxis labels / ticks
    ax.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])
      plt.savefig('results/dpgmm/dist_data/alpha_%i_%i_clusters_sil_small.png' % (num,
n_clusters))
```